DOCTOR OF SCIENCES

Hessian approximation in multilevel nonlinear optimization

Malmedy, Vincent

Award date:
2010

Awarding institution:
University of Namur

Facultés universitaires Notre-Dame de la Paix
Faculté des Sciences — Département de Mathématique

Hessian approximation in multilevel nonlinear optimization

Dissertation présentée par
Vincent Malmedy
en vue de l'obtention du grade
de *Docteur en Sciences*

Composition du jury :

Moritz Diehl
André Füzfa
Serge Gratton
Anne Lemaître (présidente)
Annick Sartenaer
Philippe L. Toint (promoteur)

2010

Imprimé en Belgique
ISBN : 978-2-87037-680-5
Dépôt légal : D / 2010 / 1881 / 25

Facultés universitaires Notre-Dame de la Paix
Faculté des Sciences
Rue de Bruxelles 61, B-5000 Namur, Belgium

Facultés universitaires Notre-Dame de la Paix
Faculté des Sciences
Rue de Bruxelles 61, B-5000 Namur, Belgium

Approximation du hessien en optimisation non linéaire multi-niveaux

Vincent Malmedy

Résumé. — Ce travail de recherche porte sur l'étude algorithmique des techniques d'approximation du hessien dans le contexte des problèmes d'optimisation non linéaire multi-niveaux. Des méthodes basées sur des approximations par différences finies ou sur des équations sécantes sont envisagées. Nous présentons de nouvelles méthodes pour approximer le hessien et comparons numériquement leurs performances par rapport à des méthodes existantes. Un logiciel implémentant une méthode efficace d'approximation du hessien dans ce contexte a été développé et est documenté dans ce document. Nous présentons également une application de ces développements à la modélisation des motifs de pigmentation sur les peaux de serpents.

Hessian approximation in multilevel nonlinear optimization

Vincent Malmedy

Abstract. — This research concerns the algorithmic study of Hessian approximation in the context of multilevel nonlinear optimization problems. Methods using finite-difference approximations or secant equations are here considered. We present new methods for the Hessian approximation and update, and numerically compare their performance with existing methods. A software implementing an efficient method for Hessian approximation in this context has been developed, and is documented in this manuscript. Moreover an application of these developments to the modelling of snake-skin pigmentation patterns is presented.

Dissertation doctorale en Sciences (orientation mathématique)
(Ph.D. thesis in Mathematics)

Date : 10 septembre 2010
Département de Mathématique
Promoteur (advisor) : Pr Ph. L. Toint

Remerciements

Cette thèse n'aurait pu voir le jour sans le concours de nombreuses personnes qui m'ont guidé, soutenu et encouragé tout au long de mon doctorat.

En premier lieu, je tiens à témoigner toute ma gratitude à Philippe Toint, mon promoteur, pour l'inestimable support qu'il m'a fourni, depuis nos premières rencontres pour esquisser un projet de thèse jusqu'à ces derniers jours pour peaufiner ce document. Merci d'abord de m'avoir accueilli au sein de l'équipe d'Analyse numérique. Merci d'avoir toujours su m'orienter sur de nouvelles pistes, tel un geyser d'idées, tout en me laissant de l'espace pour faire mes propres choix, dans un subtil mélange de guidance et d'autonomie. Merci enfin de m'avoir donné l'opportunité de participer à plusieurs conférences nationales et internationales, au cours desquelles j'ai pu présenter mes travaux et en discuter avec plusieurs chercheurs.

Je souhaite également exprimer ma reconnaissance à Serge Gratton pour les nombreux conseils qu'il m'a prodigués ainsi que pour le temps et l'énergie qu'il m'a consacrés tout particulièrement lors de mon séjour au CERFACS[1] à Toulouse. Toute ma reconnaissance s'adresse également à André Füzfa, qui a toujours marqué un vif intérêt pour mes travaux et avec qui j'ai travaillé sur une application dans le domaine de la biologie. Je les remercie tous deux pour leur enthousiasme communicatif. Je tiens également à remercier chaleureusement Dimitri Tomanos et Melissa Weber Mendonça, mes aînés dans l'équipe « multi-niveaux », pour les nombreuses discussions que nous avons partagées et pour les réponses et les conseils qu'ils m'ont donnés. Un merci tout particulier à Dimitri pour m'avoir guidé dans l'écriture de logiciels scientifiques et pour la collaboration étroite que nous avons menée en la matière. Merci enfin à Fabian Bastin, Jean-Charles Delvenne, Dominique Lambert et Mélodie Mouffe pour leurs précieuses contributions aux articles que nous avons co-écrits.

Mes remerciements vont ensuite aux membres de mon comité d'accompagnement, qui ont suivi mon travail de thèse durant ces quatre années et se sont assurés que je ne me fourvoyais point : Serge Gratton, Annick Sartenaer et Philippe Toint. Je souhaite également remercier les membres de mon jury pour leurs conseils pertinents et leurs remarques constructives ainsi que pour leur

[1] Centre Européen de Recherche et Formation Avancées en Calcul Scientifique

patience et leur persévérance dans la lecture de ce manuscrit : Moritz Diehl, André Füzfa, Serge Gratton, Anne Lemaître, Annick Sartenaer et Philippe Toint.

C'est avec une pensée toute particulière que je remercie chaleureusement celles et ceux qui m'ont accompagné au quotidien : mes collègues de bureau. Tout d'abord, merci à Caroline Sainvitu de m'avoir accueilli dans son bureau avec une grande gentillesse. Un tout grand merci ensuite à Laetitia Legrain et Patrick Laloyaux pour leur bonne humeur, pour nos conversations tantôt sérieuses tantôt frivoles ainsi que pour leurs encouragements et leur soutien constant dans la rédaction de ce manuscrit.

Pour la bonne ambiance qu'ils y font régner, je souhaite remercier tous les membres du département de Mathématique. Merci tout spécialement à mes collègues scientifiques de l'équipe d'Analyse numérique : Romain Dujol, Selime Gürol, Patrick Laloyaux, Laetitia Legrain, Margherita Porcelli, Caroline Sainvitu, Charlotte Tannier, Dimitri Tomanos, Jean Tshimanga Ilunga, Emilie Wanufelle et Melissa Weber Mendonça, notamment pour les bons moments partagés avec la plupart d'entre eux lors de conférences. Merci également à tous les autres scientifiques du département que j'ai côtoyés pendant mon doctorat, que ce soit dans le cadre académique ou extra-académique : Johan Barthelemy, Joffray Baune, Charlotte Beauthier, Jehan Boreux, Marie Castaigne, Audrey Compère, Eric Cornelis, Sandrine D'Hoedt, Martine De Vleeschouwer, Jérémy Dehaye, Nicolas Delsate, Julien Deschamps, Julien Dufey, Nicolas Franco, Valérie Goffin, Charles Hubaux, Anne-Sophie Libert, Nguyen Thi Thu Van, Benoît Noyelles, Xavier Pauly, Vincent Piefort, Simone Righi, Stéphane Valk, Emilie Verheylewegen, Fabien Walle et Sebastian Xhonneux. Merci encore aux secrétaires Murielle Haguinet, Pascale Hermans et Martine Van Caenegem pour leur dévouement jamais démenti et leur aide dans les démarches administratives.

Je souhaite par ailleurs remercier Iain Duff, Serge Gratton et Xavier Vasseur de m'avoir accueilli trois mois dans l'équipe *Parallel Algorithms* du CERFACS. Merci aux membres de cette équipe, qui m'ont rapidement intégré parmi eux et qui ont rendu encore plus agréable ce séjour sous le soleil toulousain : Audrey Bonnement, Milagros Garcia, Xueping Guo, Azzam Haidar, Mélodie Mouffe, Xavier Pinel, Léon Tham, Anke Troeltzsch, Jean Tshimanga Ilunga, Bora Ucar et François Vilar. Je tiens aussi à remercier Nicole Boutet et Suzanne Loret, qui se sont chargées des formalités administratives de ce séjour de part et d'autre de la frontière franco-belge.

Je tiens encore à exprimer ma gratitude envers le Fonds de la Recherche scientifique (FNRS) pour m'avoir soutenu financièrement pendant ces années de doctorat. De même, je souhaite remercier les Facultés universitaires Notre-Dame de la Paix de m'avoir hébergé pendant cette période et de m'avoir fourni, au sein du département de Mathématique, un environnement de travail propice au développement de ma recherche. Merci en particulier au Révérend Père Recteur Michel Scheuer, qui a soutenu mon projet de recherche auprès du FNRS et a toujours été attentif à l'avancée de celui-ci.

Mes remerciements s'adressent en outre à l'Assemblée générale des étudiants des FUNDP, pour le dépaysement qu'elle m'a procuré pendant cette thèse, tant lorsque j'y étais actif que lorsque je profitais des nombreuses activités organisées. Merci tout spécialement à ses anciens présidents Michaël Verbauwhede, Edouard Dieudonné et Arne Robbe pour les moments inoubliables que nous avons passés ensemble et pour nos discussions parfois animées. Merci enfin à Jean-François Dury et Coralie Duflocq pour leur écoute attentive et leur aide précieuse de l'autre côté du miroir.

Que tous ceux qui ont contribué à faire germer en moi le goût pour les mathématiques et la résolution de problèmes soient également remerciés. Tout d'abord, un tout grand merci à Raymond Pierret, mon professeur de mathématique dans le secondaire supérieur. Si ce passionné se refuse toujours à penser qu'il m'a fait progresser, préférant plutôt humblement dire qu'il ne m'a pas gâché, c'est pourtant grâce à lui que j'ai acquis une bonne formation mathématique de base et une démarche rigoureuse. C'est aussi suite à son engagement que j'ai pu participer à l'Olympiade mathématique belge. Ma reconnaissance s'adresse ensuite aux nombreux bénévoles qui m'ont formé pendant trois ans lors des stages de préparation à l'Olympiade mathématique internationale. Un merci tout particulier à Fabienne et Gérald Troessaert, pour leur engagement en faveur des jeunes matheux et pour la chance qu'ils m'ont offerte de découvrir d'autres aspects des mathématiques et de tisser un réseau d'amis passionnés par cette discipline.

Enfin, un profond merci à ma famille pour m'avoir entouré et soutenu jour après jour depuis toujours.

En bref, merci à tous ceux et toutes celles qui ont contribué, de près ou de loin, et parfois dans l'ombre, à l'aboutissement de cette thèse de doctorat.

Vincent

Contents

Introduction

A world involved in optimization

Optimization consists in determining the best choice from a given set of feasible alternatives. It therefore relies on three constituents: variables whose values may be chosen, potential restrictions on the allowed values of the variables, and a criterion to discriminate among the available choices. Consider for instance the situation of doing shopping where one may choose between different products. The variables are then the quantities of bought products. These quantities are nonnegative, resulting in constraints on the variables. Finally, one wishes to pay as little as possible. This determines our criterion: the total cost of the products, which we want to minimize.

Optimization arises everywhere in the real world, from natural phenomena to human life. Soap bubbles take a spherical shape, the one that minimizes the surface of contact with the outer environment for a given volume. Honeycombs minimize the wax quantity to store bee larvae. This observation has marvelled humanity from the antiquity all the more so as the regular hexagonal pattern that bees naturally create was only recently proved to be the best possible solution (Hales, 2001; see also the discussion on the topic by Hildebrandt and Tromba, 1998). The reflection of light against a mirror simply corresponds to the shortest path that touches the mirror. More generally, most of the classical physics and a part of quantum physics are ruled by the principle of least (or stationary) action (introduced by Fermat, 1662, and Maupertuis, 1746; see again Hildebrandt and Tromba, 1998).

People daily optimize without really noticing it, when doing shopping, when choosing a route, etc. But they also increasingly look for good or optimal solutions to day-to-day problems or industrial applications in a conscious way. For instance, designing lenses for people suffering from both myopia and presbyopia represents a challenging but practical problem to obtain a smooth transition between the correction areas while minimizing the astigmatism in that intermediary region (see notably Loos, Greiner and Seidel, 1997, Toint and Tomanos, 2009, and Tomanos, 2009). In the food processing industry, one aims at keeping a maximum of nutrients while reducing the number of bacteria under a legal

threshold by heating the packaged food in steam or hot water autoclaves (see Sachs, 2003). Currently, weather forecast is elaborated with data assimilation techniques, in which the deviation between the mathematical weather model and the past observations is minimized (see for instance Fisher, 1998). The aeronautical structure design is another field in which optimization is highly present; manufacturers indeed always wish to minimize the plane weight while maintaining its structural integrity. In video games, shock simulation also calls for the solution of optimization problems in real time for fluid video animation (see notably Anitescu and Potra, 1996). Optimization is furthermore involved in the finance environment for risk management, portfolio analysis, etc.

Practical examples of optimization are so numerous and varied that we could hardly exhaust this listing. To sketch a broad scope of its applications, let us say that optimization mainly covers problems in physics, chemistry, biology, medicine, planning, industry, economics, statistics and social management.

Taxonomy

Over the years, mathematicians have designed methods to solve these optimization problems. In their quest for efficiency (and robustness), they have attempted to exploit as much as possible special features of the considered problems, and this thesis is no exception to the rule. As a consequence, optimizations problems are divided into several categories.

The most common taxonomy derives from the three constituents of optimization problems. First, we distinguish the problems on the nature of their variables: coming from either a continuous space (typically the set of real numbers), or a discrete space (typically the set of integer numbers), yielding the fields of *continuous* and *discrete* optimization, respectively. The second distinction is based on the existence of constraints on the variables, yielding the fields of *unconstrained* and *constrained* optimization. Finally, the nature of the objective function that we optimize (and of the constraints) separates the fields of *linear* and *nonlinear* optimization (depending on whether the objective function and constraints are linear or not). Further classifications based on properties of the objective function and constraints are mentioned in the literature (see for instance Chapter 1 in Nocedal and Wright, 2006).

For historical reasons, one also referred to optimization as *mathematical programming*, though it does not refer to computer programming. In particular, the name *linear programming* (proposed by Dantzig in the 1940s) is frequently used instead of *linear optimization*. In such cases, one usually speaks of *programs* instead of *problems*.

This thesis mainly takes place in the context of continuous nonlinear unconstrained optimization, though some of the presented methods are related to linear and/or constrained optimization. Discrete optimization is never consid-

ered in what follows. The two next sections refine the definition of our research object.

Multilevel optimization

Many phenomena are nowadays modelled with partial differential equations (PDEs). The variables are then functions, meaning that an infinite number of unknowns should be determined. Such a task often turns out to be impractical, so one commonly only determines an approximation of the true solution by discretizing the problem, that is by considering it only on (a potentially large number of) points of the function domain. The more points are used, the better one expects the approximate solution to represent the true solution. In that process, one may obviously select many different sets of discretization points, hence leading to more or less faithful representations of the original problem. These ideas are formalized in the notion of *multilevel problems*, which are defined with different levels of accuracy (for instance, on grids with increasing mesh size). The development and study of methods efficiently dealing with such problems is the central topic of this thesis.

Hessian approximation

The Hessian matrix, which contains the curvature information on the objective function, plays a major role in continuous optimization since the seminal Newton's method (see Section 2.3). Assuming that n variables are used to describe a problem, this matrix has n^2 entries. Large-scale problems may hence lead to intractable Hessian, either in computation time, or in memory storage. The Hessian therefore needs to be approximated to counteract these drawbacks. In the context of multilevel problems, several properties of the objective functions may then be exploited, as well as multilevel hierarchy itself, to this purpose. The topic of this thesis is therefore the Hessian approximation in nonlinear multilevel optimization.

Structure of the document and contributions

The thesis is divided in three parts: first, prolegomena that properly state the framework of multilevel optimization methods (Part I); then a discussion on Hessian approximation (Part II); finally, the presentation of an application and software (Part III).

Starting the prolegomena, Chapter 1 states some notations and recalls basic concepts of linear algebra, topology, analysis and graph theory. It also introduces a tool for benchmarking algorithms. In Chapter 2, we present fundamentals of nonlinear optimization, starting from the characterization of the

solutions and then presenting iterative methods to compute them. Newton's method is hence described first as well as its alter ego: the quasi-Newton methods, which use an approximate Hessian instead of the exact Hessian. Since these methods can fail to converge from some starting points, we next consider globalization techniques: linesearch methods, and in particular conjugate gradient methods, as well as trust-region methods, so ensuring the convergence to a solution irrespective of the starting point. We conclude this chapter by our first contribution: the retrospective trust-region method, whose convergence theory is established, and whose numerical performance is investigated. Chapter 3 introduces the multilevel methods, first in the context of linear systems of equations, and then in that of nonlinear optimization, yielding the recursive multilevel trust-region (RMTR) method. This chapter has a pivotal role in the thesis, since the RMTR method is used as optimization solver in the numerical experiments of Chapters 5 and 7, and since the multilevel principles highlighted in this chapter are also at the basis of the developments in Chapter 6. This concludes the prolegomena.

Part II, which is devoted to Hessian approximation, is also divided in three chapters, starting from an introduction to this matter in Chapter 4. Therein we classify the potential techniques for Hessian approximation in three categories: finite-difference methods, secant methods (which are based on secant equations), and automatic differentiation methods. We then consider in Chapter 5 the design of Hessian approximations for problems arising from a discretization, and which therefore typically present a sparse Hessian matrix and are partially separable. We first review two existing methods for sparse Hessian approximation, one from the finite-difference type, and the other from the secant type. We then propose a secant Hessian approximation for partially separable functions, and attempt afterwards to enforce the positive definiteness of the approximation. Numerical experiments are conducted to compare the considered methods. These new methods and comparative experiments constitute our second contribution. Finally, Chapter 6 focuses on a limited-memory Hessian approximation whose construction relies on the multilevel hierarchy of the considered problem. After a brief introduction to the existing L-BFGS method, we describe a theoretical framework in which more than one secant equation is generated at each iteration. Extensive numerical experiments are conducted first to compare many variants using the newly generated secant equations, and second to understand the increased performance of the new method with respect to the L-BFGS method. An alternative to the L-BFGS Hessian approximation is eventually proposed. These numerical experiments and the alternative approximation method constitute our third contribution.

Finally, Part III contains two chapters dedicated to an application and to software, respectively. Chapter 7, which constitutes our fourth contribution, presents an application of the RMTR method using Hessian approximations to a subject in mathematical biology: the modelling of snake-skin pigmentation patterns. We first describe the problem modelling from the biological aspect to

the discretization aspect. Numerical solutions of the resulting least-squares problem are then given, and the solution efficiency is discussed. We then elaborate on two techniques to characterize the generated patterns, namely Fourier analysis and random walks on discrete graphs. The development of a finite-element discretization is finally considered. Chapter 8 describes the two software packages (RMTR and LTS) what were partly or completely developed during this thesis. While their specifications are given in Appendices D and E, respectively, we yet focus in that chapter on a few notable aspects of these software packages, namely the predefined transfer operators and Hessian management in RMTR, and the predefined sparsity patterns in LTS.

Part I

Prolegomena

Chapter 1

Basic concepts

*Dīcēbăt Bērnārdŭs Cārnotēnsĭs nōs ēssĕ quăsĭ nānōs, gĭgāntĭum hŭmĕrīs īnsīdēntēs, ŭt pōssīmŭs plūră ĕīs ĕt rĕmōtĭōră vīdĕrĕ, nōn utīquĕ prŏprĭī vīsūs ăcūmĭnĕ, āūt ēmĭnēntĭā cōrpŏrĭs, sĕd quĭă ĭn āltum sūbvĕnīmŭr ĕt ēxtōllĭmŭr māgnĭtūdĭnĕ gĭgāntĕā.**

This preliminary chapter aims to state some notations (see also the list of notations on pages 207 and following) and recall basic concepts that are used in the following chapters. We hence review some elements of linear algebra in Section 1.1, of topology in Section 1.2, of mathematical analysis in Section 1.3, and of graph theory in Section 1.4. We finally present in Section 1.5 a convenient tool to compare performances of methods.

Note that $|\mathcal{S}|$ denotes the cardinality of the set $\mathcal{S}$, that is the number of elements that it contains.

1.1 Elements of linear algebra

1.1.1 Vectors

Given a positive integer n, we denote by $\mathbb{R}^n$ the real n-dimensional Euclidean space, which is a vector space over the set $\mathbb{R}$ of real numbers. Each element x

** Bernard of Chartres used to say that we are like dwarfs standing on the shoulders of giants, such that we can see more things and further away than they could. And this, not because our sight would be more powerful or our height more advantageous, but because we are carried and heightened by the high stature of the giants. (John of Salisbury, 1159)

of $\mathbb{R}^n$ is an n-dimensional column *vector* of the form

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix}, \tag{1.1}$$

where $x_i \in \mathbb{R}$ $(i = 1, \ldots, n)$ is called the i-th *component* of the vector x. We denote by x^T the *transpose* of vector x, that is the n-dimensional row vector

$$x^T = \begin{pmatrix} x_1 & x_2 & \cdots & x_{n-1} & x_n \end{pmatrix}. \tag{1.2}$$

Inner product. — Considering two vectors x and y of $\mathbb{R}^n$, we denote their *inner product* by

$$\langle x, y \rangle \stackrel{\text{def}}{=} x^T y = \sum_{i=1}^{n} x_i y_i. \tag{1.3}$$

Norms. — A *norm* on a vector space $\mathcal{S}$ is a function $\|\cdot\| : \mathcal{S} \to \mathbb{R}^+$ that has the following properties:

(i) $\forall\, \alpha \in \mathbb{R}, \forall\, x \in \mathcal{S} : \|\alpha x\| = |\alpha| \|x\|$. [*positive homogeneity*]

(ii) $\forall\, x, y \in \mathcal{S} : \|x + y\| \leqslant \|x\| + \|y\|$. [*triangle inequality*]

(iii) $\|x\| = 0$ if and only if $x = 0$. [*positive definiteness*]

The most commonly used norm on $\mathbb{R}^n$ is the *Euclidean norm*, which covers the usual notion of distance, and is derived from the inner product on $\mathbb{R}^n$; it is defined by

$$\|x\|_2 \stackrel{\text{def}}{=} \sqrt{\langle x, x \rangle}. \tag{1.4}$$

This norm belongs to the larger class of ℓ_p-norms (or simply p-norms, or Hölder norms), which are defined by

$$\|x\|_p \stackrel{\text{def}}{=} \sqrt[p]{\sum_{i=1}^{n} |x_i|^p} \tag{1.5}$$

for any real $p \geqslant 1$. Another practical norm is the *infinity norm* defined as

$$\|x\|_\infty \stackrel{\text{def}}{=} \max_{1 \leqslant i \leqslant n} |x_i|. \tag{1.6}$$

All norms defined on $\mathbb{R}^n$ are *equivalent* in the sense that for any pair of such norms, each one is bounded below and above by a constant multiple of the other one.

Interestingly, the Euclidean norm satisfies the *Cauchy-Schwarz inequality*

$$|\langle x, y\rangle| \leqslant \|x\|_2 \|y\|_2. \tag{1.7}$$

Note also, that for sake of readability, the notation $\|x\|$ represents the Euclidean norm of the vector x in what follows, unless stated otherwise.

1.1.2 Matrices

Given two positive numbers m and n, we denote by $\mathbb{R}^{m\times n}$ the set of m-by-n real matrices (meaning that they have m rows and n columns of real entries). We denote by A_{ij} (for $i = 1, \ldots, m$ and $j = 1, \ldots, n$) the entry (i, j) of the matrix $A \in \mathbb{R}^{m\times n}$, that is its entry on row i and column j. We denote by A^T, the *transpose* of matrix A, that is the matrix of $\mathbb{R}^{n\times m}$ such that $[A^T]_{ij} = A_{ji}$. The *trace* of a square matrix $A \in \mathbb{R}^{n\times n}$ is the real number

$$\operatorname{tr} A \stackrel{\text{def}}{=} \sum_{i=1}^{n} A_{ii}. \tag{1.8}$$

Matrix types. — A matrix A that is equal to its own transpose is *symmetric*. Note that such a matrix must be a *square* matrix, which has the same number of rows and columns n. A symmetric matrix $A \in \mathbb{R}^{n\times n}$ is *positive semidefinite* (noted $A \succcurlyeq 0$) if

$$\langle x, Ax\rangle \geqslant 0$$

for every $x \in \mathbb{R}^n$, and is moreover *positive definite* (noted $A \succ 0$) if

$$\langle x, Ax\rangle > 0$$

for every $x \in \mathbb{R}^n \setminus \{0\}$. Given a symmetric positive definite $A \in \mathbb{R}^{n\times n}$, we can define the (ellipsoidal) A-norm on $\mathbb{R}^n$ as

$$\|x\|_A = \sqrt{\langle x, Ax\rangle}. \tag{1.9}$$

Products. — We now introduce three different product operations defined on matrices. First, the (usual) product of a matrix $A \in \mathbb{R}^{m\times p}$ by a matrix $B \in \mathbb{R}^{p\times n}$ is the matrix $AB \in \mathbb{R}^{m\times n}$ whose entries are defined by

$$[AB]_{ij} = \sum_{k=1}^{p} A_{ik} B_{kj} \tag{1.10}$$

for $i = 1, \ldots, m$ and $j = 1, \ldots, n$. The neutral element for this operation is the *identity matrix* $I_n \in \mathbb{R}^{n\times n}$ (or simply I, if there is no confusion on the dimension), whose entries are

$$I_{ij} = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{if } i \neq j, \end{cases} \tag{1.11}$$

for $i, j = 1, \ldots, n$. The inverse of a square matrix A of $\mathbb{R}^{n\times n}$ is the unique matrix A^{-1} of $\mathbb{R}^{n\times n}$ such that

$$AA^{-1} = I = A^{-1}A. \tag{1.12}$$

Note also that a square matrix A of $\mathbb{R}^{n\times n}$ is *orthogonal* if and only if

$$A^T A = I, \tag{1.13}$$

and is *normal* if and only if

$$A^T A = AA^T. \tag{1.14}$$

Second, the Hadamard (or entrywise) product of two matrices $A, B \in \mathbb{R}^{m\times n}$ is the matrix $A \bullet B \in \mathbb{R}^{m\times n}$ whose entries are defined by

$$[A \bullet B]_{ij} = A_{ij}B_{ij} \tag{1.15}$$

for $i = 1, \ldots, m$ and $j = 1, \ldots, n$. The neutral element for this operation is the matrix $J_{m,n}$ (or J if no confusion is possible), whose entries are all 1. The Hadamard-inverse of a matrix A of $\mathbb{R}^{m\times n}$ is the unique matrix $A^{\Hadamard}$ of $\mathbb{R}^{m\times n}$ such that $A \bullet A^{\Hadamard} = J_{m,n} = A^{\Hadamard} \bullet A$, yielding

$$[A^{\Hadamard}]_{ij} = (A_{ij})^{-1}. \tag{1.16}$$

More details on this product may be found in Chapter 5 of Horn and Johnson (1985).

Finally, the Kronecker product of a matrix $A \in \mathbb{R}^{m\times n}$ by a matrix $B \in \mathbb{R}^{p\times q}$ is the matrix $A \otimes B \in \mathbb{R}^{mp\times nq}$ defined by

$$A \otimes B = \begin{pmatrix} A_{11}B & \cdots & A_{1n}B \\ \vdots & \ddots & \vdots \\ A_{m1}B & \cdots & A_{mn}B \end{pmatrix}. \tag{1.17}$$

This product is often used in conjunction with the $\mathrm{vec}(\cdot)$ operator which converts matrices into vectors, and is defined by

$$\mathrm{vec}(A) = (A_{11}, \ldots, A_{m1}, A_{12}, \ldots, A_{m2}, \ldots, A_{1n}, \ldots, A_{mn}). \tag{1.18}$$

We then have the interesting relation

$$\mathrm{vec}(ABC) = (C^T \otimes A)\,\mathrm{vec}\,B, \tag{1.19}$$

yielding in particular $\mathrm{vec}(AB) = (I \otimes A)\,\mathrm{vec}\,B = (B^T \otimes I)\,\mathrm{vec}\,A$. Again, Horn and Johnson (1985, Chapter 4) describe in more detail this product.

Eigenvalues, singular values, and invariant subspaces. — An *eigenvalue* of a square matrix $A \in \mathbb{R}^{n\times n}$ is a scalar λ such that

$$Ax = \lambda x \tag{1.20}$$

for some vector x, called an *eigenvector* associated to the eigenvalue λ. The set of the eigenvalues of a matrix is its *spectrum*, and we denote by $\rho(A)$, the *spectral radius* of A, that is the maximum of the absolute values of the eigenvalues. Each symmetric matrix of $\mathbb{R}^{n\times n}$ has n real eigenvalues. A positive semidefinite matrix has only nonnegative eigenvalues, while a positive definite matrix has only positive eigenvalues.

The notion of invariant subspace extends that of eigenvector. More precisely, the vector space $\mathcal{S} \subset \mathbb{R}^n$ is an *invariant subspace* of the matrix $A \in \mathbb{R}^{n\times n}$ if $Ax \in \mathcal{S}$ for each $x \in \mathcal{S}$. In particular, the space spanned by each eigenvector of A is an invariant subspace of A.

The singular values of the matrix $A \in \mathbb{R}^{n\times n}$ are the square roots of the eigenvalues of A^TA. Observe that singular values and eigenvalues coincide when the matrix is symmetric positive semidefinite. We denote the smallest one by $\sigma_{\min}(A)$, and the largest one by $\sigma_{\max}(A)$. Finally, the *condition number* of the matrix A is the ratio $\kappa(A) \stackrel{\text{def}}{=} \sigma_{\max}(A)/\sigma_{\min}(A)$ (see Golub and Van Loan, 1996, for more precision).

Norms. — A matrix norm $\|\cdot\|_M$ on $\mathbb{R}^{m\times n}$ may be *induced* by two vector norms ($\|\cdot\|_U$ on $\mathbb{R}^m$, and $\|\cdot\|_V$ on $\mathbb{R}^n$) in the sense that it is defined by

$$\|A\|_M = \max_{x\neq 0} \frac{\|Ax\|_U}{\|x\|_V} \tag{1.21}$$

for every matrix $A \in \mathbb{R}^{m\times n}$. In particular, the matrix norms induced by the vector 1-norm, 2-norm and infinity norm are

$$\|A\|_1 = \max_{1\leqslant j\leqslant n} \sum_{i=1}^{m} |a_{ij}|, \quad \|A\|_2 = \sigma_{\max}(A), \quad \text{and} \quad \|A\|_\infty = \max_{1\leqslant i\leqslant m} \sum_{j=1}^{n} |a_{ij}|$$

respectively. Some common matrix norms are not induced, as the entrywise *Frobenius norm*

$$\|A\|_F \stackrel{\text{def}}{=} \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} A_{ij}^2}. \tag{1.22}$$

This norm is also sometimes referred as the Euclidean matrix norm since it is derived from the common *matrix inner product* defined by

$$\langle A, B\rangle = \operatorname{tr}(A^TB) \tag{1.23}$$

for all matrices $A, B \in \mathbb{R}^{m\times n}$; hence, $\|A\|_F = \sqrt{\langle A, A\rangle}$.

In the case of square matrices belonging to $\mathbb{R}^{n\times n}$, we may also require a matrix norm to be *consistent*, in the sense that it additionally satisfies the fourth property:

(iv) $\forall\, A, B \in \mathbb{R}^{n\times n} : \|AB\| \leqslant \|A\|\|B\|$. [*submultiplicativity*]

The matrix 1-norm, 2-norm, infinity norm and Frobenius norm are all consistent.

1.2 Elements of topology

We here briefly present some elements of general topology for the space $\mathbb{R}^n$ with its usual topology. We define the *ball* of radius ϵ centred at x as

$$\mathcal{B}_\epsilon(x) \stackrel{\text{def}}{=} \{y \in \mathbb{R}^n : \|y - x\| < \epsilon\}\,. \tag{1.24}$$

Any set $\mathcal{N}$ containing a ball centred at the point $x \in \mathbb{R}^n$ (of whatever positive radius) is a *neighbourhood* of x. A subset $\mathcal{S}$ of $\mathbb{R}^n$ is *open* if for each point $x \in \mathcal{S}$, there exists a positive constant ϵ such that $\mathcal{B}_\epsilon(x) \subset \mathcal{S}$. A subset $\mathcal{S}$ of $\mathbb{R}^n$ is *closed* if its complement in $\mathbb{R}^n$ — that is $\mathbb{R}^n \setminus \mathcal{S}$ — is open.

The *interior* of a set $\mathcal{S}$ is the largest (in the sense of set inclusion) open set that is contained in $\mathcal{S}$, or equivalently the union of every open set contained in $\mathcal{S}$. The *closure* of a set $\mathcal{S}$ is the smallest (in the sense of set inclusion) closed set that contains $\mathcal{S}$, or equivalently the intersection of all closed set that contain $\mathcal{S}$. We then denote by $\partial\mathcal{S}$ the *boundary* of the set $\mathcal{S}$, that is the difference between the closure and the interior of the set $\mathcal{S}$.

A subset $\mathcal{S}$ of $\mathbb{R}^n$ is *bounded* if there exists a positive constant K such that $\|x\| \leqslant K$ for all $x \in \mathcal{S}$. A subset $\mathcal{S}$ of $\mathbb{R}^n$ is *compact* if and only if it is both closed and bounded.

Given two points $x, y \in \mathbb{R}^n$, we may define the *segment*

$$[x, y] \stackrel{\text{def}}{=} \{\lambda x + (1 - \lambda)y : \lambda \in [0, 1]\} \tag{1.25}$$

that joins these two points. A set $\mathcal{S}$ is *convex* if and only if $[x, y] \subset \mathcal{S}$ for all points $x, y \in \mathcal{S}$.

1.3 Elements of analysis

We first recall in Subsection 1.3.1 when a sequence of iterates converges and how to characterize the speed of this convergence. We then describe in Subsection 1.3.2 some useful properties that a function may have, and devote more particularly Subsection 1.3.3 to its differentiability properties. Finally, we introduce briefly partial differential equations in Subsection 1.3.4.

1.3.1 Convergence of sequences

In iterative procedures generating a sequence of vectors x, we refer to the k-th iterate as x_k, while its i-th component is then denoted by $[x_k]_i$. The sequence $\{x_k\} \subset \mathbb{R}^n$ *converges* to the vector $x_* \in \mathbb{R}^n$ if

$$\forall\, \epsilon > 0, \exists\, K \in \mathbb{N}, \forall\, k \geqslant K : \|x_k - x_*\| < \epsilon, \tag{1.26}$$

and we denote it by

$$x_k \to x_* \;\text{ or }\; \lim_{k\to\infty} x_k = x_*, \tag{1.27}$$

saying that x_* is the *limit* of the sequence $\{x_k\}$. In contrast, a point x^* is a *limit point* (or *accumulation point*) of the sequence $\{x_k\}$ if there exists a subsequence $\{x_k\}_{k\in\mathcal{K}}$ ($\mathcal{K} \subset \mathbb{N}$) that converges to x^*. When dealing with sequences of real numbers, the concepts of *limit superior* and *limit inferior* refer to the largest and smallest limit point of the sequence, respectively. More formally, we define them as

$$\limsup_{k\to\infty} x_k = \inf_{k\geqslant 0} \sup_{\ell\geqslant k} x_\ell \qquad \text{and} \qquad \liminf_{k\to\infty} x_k = \sup_{k\geqslant 0} \inf_{\ell\geqslant k} x_\ell, \tag{1.28}$$

respectively.

Rates of convergence. — When an iterative procedure converges, the iterates x_k may get closer to the limit point x_* more or less quickly. This idea is formalized in the concept of *rate of convergence* (see Chapter 9 in Ortega and Rheinboldt, 1970, for a thorough discussion). While other characterizations of the convergence speed exist, we here only present *quotient rates of convergence* (or Q-rates).

The sequence $\{x_k\}$ converges *Q-linearly* if

$$\limsup_{k\to\infty} \frac{\|x_{k+1} - x_*\|}{\|x_k - x_*\|} = \mu \tag{1.29}$$

for some $\mu \in (0, 1)$. It converges *Q-superlinearly* if

$$\lim_{k\to\infty} \frac{\|x_{k+1} - x_*\|}{\|x_k - x_*\|} = 0 \tag{1.30}$$

Finally, it converges *with Q-order* $r > 1$ if

$$\limsup_{k\to\infty} \frac{\|x_{k+1} - x_*\|}{\|x_k - x_*\|^r} \tag{1.31}$$

has a nonnegative finite value. In these expressions, the quotient is assumed to be zero if $x_k = x_{k+1} = x_*$. Convergence with Q-order 2 is called *Q-quadratic* convergence. For the sake of simplicity, we drop the Q prefix in what follows.

Local vs. global convergence. — Many iterative procedures only define how an iterate x_k is built from the former one(s), but do not specify the starting point x_0, letting entire freedom for this choice to the user. We would therefore like to distinguish methods that are robust enough to converge starting from whatever point. Mathematically, an iterative method that converges to a solution irrespective of the starting point is *globally convergent*. If it converges only when starting from some points (typically close enough to the solution), it is *locally convergent*.

1.3.2 Some function characteristics

In this subsection, we consider a function $f : \mathcal{D} \to \mathbb{R}$ defined on an open subset $\mathcal{D}$ of $\mathbb{R}^n$ (potentially $\mathbb{R}^n$ itself), and describe some useful properties that this function may have.

Convexity. — Given a convex domain $\mathcal{D}$, the function f is *convex* if

$$f(\lambda x + (1-\lambda)y) \leqslant \lambda f(x) + (1-\lambda) f(y) \tag{1.32}$$

for all $x, y \in \mathcal{D}$ and $\lambda \in [0, 1]$.

Partial separability. — A function f is *partially separable* if there exists a decomposition of the form

$$f(x) = \sum_{i=1}^{m} f_i(x), \tag{1.33}$$

where each *element function* $f_i : \mathcal{D} \to \mathbb{R}$ $(i = 1, \ldots, m)$ depends only on a few variables, say those in some subset $\mathcal{I}_i \subset \{1, \ldots, n\}$ with $|\mathcal{I}_i| = n_i \ll n$. The concept of partial separability was introduced by Griewank and Toint (1982*b*). While the original concept is slightly more general, we will consider only this definition in what follows.

Continuity. — The function f is *continuous* at the point x if

$$\lim_{y \to x} f(y) = f(x). \tag{1.34}$$

A function f is continuous if it is continuous at each point of its domain $\mathcal{D}$.

Lipschitz continuity. — A function $f : \mathcal{D} \to \mathbb{R}$ is *Lipschitz continuous* if there exists a constant $L > 0$ such that for all $x, y \in \mathcal{D}$,

$$\|f(x) - f(y)\| \leqslant L\|x - y\|. \tag{1.35}$$

Every Lipschitz continuous function is continuous. The smallest constant L for which the condition (1.35) holds is called the *Lipschitz constant* of the function f.

1.3.3 Derivatives

First derivatives. — Consider first the univariate function $f : \mathcal{D} \to \mathbb{R}$ defined on an open subset $\mathcal{D}$ of $\mathbb{R}$. This function is *differentiable* at the point x if the limit

$$f'(x) \stackrel{\text{def}}{=} \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \tag{1.36}$$

exists in $\mathbb{R}$. In this case, $f'(x)$ is called the (first) derivative of f at x, which is also denoted by $\frac{df}{dx}$. If the function f depends on a variable y that is itself a function of the variable x, then the *chain rule* formulates that

$$\frac{df}{dx} = \frac{df}{dy}\frac{dy}{dx}. \tag{1.37}$$

Now, let $\mathcal{D}$ be an open subset of $\mathbb{R}^n$, and $f : \mathcal{D} \to \mathbb{R}$ be a multivariate function. The *(first) partial derivative* of f with respect to the variable x_i $(i = 1, \ldots, n)$ is

$$\frac{\partial f(x)}{\partial x_i} = \lim_{h \to 0} \frac{f(x + he_i) - f(x)}{h} \tag{1.38}$$

where e_i is the i-th unit vector, and the gradient of f at x is

$$\nabla f(x) = \begin{pmatrix} \frac{\partial f(x)}{\partial x_1} \\ \vdots \\ \frac{\partial f(x)}{\partial x_1} \end{pmatrix}. \tag{1.39}$$

The function f is *differentiable at* x if and only if

$$\lim_{y \to x} \frac{f(y) - f(x) - \langle \nabla f(x), y - x \rangle}{\|y - x\|} = 0. \tag{1.40}$$

The function f is *differentiable* if it is differentiable at each point of its domain $\mathcal{D}$. If moreover the partial derivatives of f are continuous, the function f is *continuously differentiable*.

Second derivatives. — The second partial derivatives of the function f are

$$\frac{\partial^2 f(x)}{\partial x_i \partial x_j} = \frac{\partial}{\partial x_i}\left(\frac{\partial f(x)}{\partial x_j}\right) \tag{1.41}$$

for $i, j = 1, \ldots, n$. These values are gathered in the *Hessian* matrix

$$\nabla^2 f(x) = \begin{pmatrix} \frac{\partial^2 f(x)}{\partial x_1^2} & \frac{\partial^2 f(x)}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(x)}{\partial x_2 \partial x_1} & \frac{\partial^2 f(x)}{\partial x_2^2} & \cdots & \frac{\partial^2 f(x)}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(x)}{\partial x_n \partial x_1} & \frac{\partial^2 f(x)}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_n^2} \end{pmatrix}. \tag{1.42}$$

If the second partial derivatives of the function f are continuous, this function is *twice-continuously differentiable*, and its Hessian matrix is symmetric by virtue of Schwarz's theorem.

Mean-value theorem. — We conclude this section on mathematical analysis with a useful result for optimization theory.

Theorem 1.1 (Mean-value theorem) *Let $\mathcal{D}$ be an open subset of $\mathbb{R}^n$, and assume that the function $f : \mathcal{D} \to \mathbb{R}$ is continuously differentiable through $\mathcal{D}$. Then if the segment $[x, x+s]$ lies in $\mathcal{D}$,*

$$f(x+s) = f(x) + \langle \nabla f(x+\alpha s), s \rangle \,, \tag{1.43}$$

for some $\alpha \in (0,1)$. Moreover, if the function f is twice-continuously differentiable through $\mathcal{D}$, then

$$\nabla f(x+s) = \nabla f(x) + \int_0^1 \nabla^2 f(x+\hat{\alpha} s) s \, d\alpha, \tag{1.44}$$

and

$$f(x+s) = f(x) + \langle \nabla f(x), s \rangle + \tfrac{1}{2} \left\langle s, \nabla^2 f(x+\check{\alpha} s) s \right\rangle \tag{1.45}$$

for some $\hat{\alpha}, \check{\alpha} \in [0,1]$.

Due to this theorem, we may approximate the function f in a neighbourhood of any point x of its domain by its linear Taylor approximation

$$m(x+s) = f(x) + \langle \nabla f(x), s \rangle \,, \tag{1.46}$$

provided that the function f is continuously differentiable, and by its quadratic Taylor approximation

$$m(x+s) = f(x) + \langle \nabla f(x), s \rangle + \tfrac{1}{2} \left\langle s, \nabla^2 f(x) s \right\rangle \,, \tag{1.47}$$

provided that the function f is twice-continuously differentiable.

1.3.4 Partial differential equations

A *partial differential equation* (PDE) is some relation involving an unknown function $u : \mathcal{D} \to \mathbb{R}$ and its partial derivatives, where $\mathcal{D}$ is some open subset of $\mathbb{R}^n$. The largest order of these derivatives determines the *order* of the PDE. A partial differential equations is *linear* if the relation between the unknown function and its derivatives is linear. When the unknown function u depends

on a single variable ($n = 1$), one speaks instead of an *ordinary differential equation* (ODE).

The unknown function u is also typically required to fulfil some additional conditions. *Boundary conditions* determine the function value or some (parts) of its derivatives on the boundary $\partial\mathcal{D}$ (yielding a *boundary value problem*). By contrast, *initial conditions* give that same kind of information, but for a given value of some variable (typically defining the state of the system at the initial time). Conditions fixing the function value are called *Dirichlet conditions*, whereas *Neumann conditions* concern the first derivatives of the unknown function.

Classification. — The taxonomy of PDEs may be a complex topic, but a first well-known classification is based on linear second-order PDEs with constant coefficients, which are of the form

$$\sum_{i,j=1}^{n} a_{ij} \frac{\partial^2 u}{\partial x_i \partial x_j} + \sum_{j}^{n} b_j \frac{\partial u}{\partial x_j} + cu = f, \tag{1.48}$$

for some real constants $a_{i,j}$, b_j and c, and where f is some function from $\mathbb{R}^n$ to $\mathbb{R}$. These PDEs may be classified according to the eigenvalues of the (symmetric) coefficient matrix A (whose entries are a_{ij}):

- **Elliptic:** all the eigenvalues are nonzero and have the same sign.
- **Parabolic:** all the eigenvalues are nonzero and have the same sign, except one that is zero.
- **Hyperbolic:** all the eigenvalues are nonzero and have the same sign, except one that has the opposite sign.

Other sign combinations may occur (when $n > 2$), but are out of the scope of this brief introduction. The prototype of elliptic PDEs is the Laplace's equation

$$\Delta u \stackrel{\text{def}}{=} \sum_{j=1}^{n} \frac{\partial^2 u}{\partial x_j^2} = 0.$$

where Δ is called the *Laplace operator* or *Laplacian*. The prototype of parabolic PDEs is the heat equation

$$\frac{\partial u}{\partial t} - \Delta u = 0,$$

where the unknown function is now $u : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R} : (x, t) \mapsto u(x, t)$, and that of hyperbolic PDEs is the wave equation

$$\Box u \stackrel{\text{def}}{=} \frac{\partial^2 u}{\partial t^2} - \Delta u = 0,$$

where $\Box$ is called the *d'Alembert operator*.

1.4 Elements of graph theory

This section is solely devoted to the introduction of the push-relabel algorithm introduced by Goldberg and Tarjan (1988) to maximize flow on a network, which is used in Section 5.4. We therefore do not elaborate much on other elements of graph theory, but refer to the good introduction on the topic by Bondy and Murty (1976). First we introduce the concepts of flow and network in Subsection 1.4.1, and then present the push-relabel method in Subsection 1.4.2.

1.4.1 Flow and network

Given a set of nodes V, an *edge* is a unordered pair $v - w$ linking the nodes $v, w \in V$. By contrast, an *arc* is an ordered pair $v \to w$ going from node $v \in V$ to node $w \in V$. An (undirected) *graph* consists of a set of nodes and a set of edges, whereas a *directed graph* (or digraph) consists of a set of nodes and a set of arcs. A *network* is simply a directed graph on whose arcs a capacity function is defined.

A *flow network* is a directed graph (V, E, s, t, u) consisting of a node set V, an arc set E, a source $s \in V$, a sink $t \in V$, and a capacity function $u : E \to \mathbb{R}^+$. Let $n = |V|$ and $m = |E|$. A *flow* is a function $f : E \to \mathbb{R}^+$ that satisfies both the *capacity constraints*

$$u_f(v \to w) \stackrel{\text{def}}{=} u(v \to w) - f(v \to w) \geqslant 0, \tag{1.49}$$

for every $(v \to w) \in E$, and the *conservation constraints*

$$e_f(v) \stackrel{\text{def}}{=} \sum_{w : v \in E(w)} f(w \to v) - \sum_{w \in E(v)} f(v \to w) = 0, \tag{1.50}$$

for every $v \in V \setminus \{s, t\}$, and where $E(v) = \{w \in V : (v \to w) \in E\}$ contains the termination of the arcs starting at v. We call $u_f(v \to w)$ the *residual capacity* on arc $v \to w$, and $e_f(v)$ the *excess* of flow at node v. A *preflow* is a function $f : E \to \mathbb{R}^+$ that still satisfies (1.49) for every arc of E, but only the relaxed conservation constraints $e_f(v) \geqslant 0$ for every node of V.

An arc $v \to w$ is *saturated* if $u_f(v \to w) = 0$, and *residual* otherwise. The subset $E_f(v)$ contains the residual arcs of $E(v)$. On this flow network, we may also define a *distance labelling*, that is a function $d : V \to \mathbb{N}$ such that $d(t) = 0$ and $d(v) \leqslant d(w) + 1$ for every arc $v \to w$ (this labelling is not unique yet). A residual arc $v \to w$ is *admissible* if $d(v) = d(w) + 1$. Finally, a node v is *active* if $e_f(v) > 0$, $v \notin \{s, t\}$ and $d(v) < n$.

1.4.2 Push-relabel method

Several methods are designed to compute a maximal flow on a network (with fixed capacities), that is to maximize

$$\sum_{v:t\in E(v)} f(v \to t) \tag{1.51}$$

under the constraints (1.49) and (1.50) (see Ahuja, Magnanti and Orlin, 1993). Note that this is a linear program. Although it can thus be solved by general methods of linear programming, we consider here methods from graph theory, and focus in particular on the push-relabel method of Goldberg and Tarjan (1988), which does not require the flow to take integer value. This method indeed shows good numerical performance compared to other algorithms aiming at solving the same problem (see Ahuja, Kodialam, Mishra and Orlin, 1997). Our description of the method is inspired by Cherkassky and Goldberg (1997).

The push-relabel method is iterative and updates at each iteration k a preflow f_k and a distance labelling d_k, by performing one of the two basic operations of the method, namely the push and the relabel operations. We start our presentation of the algorithm by describing these operations, then show how the preflow f_0 and the distance labelling d_0 are initialized, and finally explain in which order the aforementioned operations are performed to update f_k and d_k.

Push and relabel operations. — Given an active node v and an admissible arc $v \to w$, the *push* operation sends $\delta = \min[e_{f_k}(v), u_{f_k}(v \to w)]$ units of flow from v to w, yielding the changes

$$f_{k+1}(v \to w) = f_k(v \to w) + \delta, \qquad e_{f_{k+1}}(w) = e_{f_k}(w) + \delta, \tag{1.52a}$$

$$u_{f_{k+1}}(v \to w) = u_{f_k}(v \to w) - \delta, \qquad e_{f_{k+1}}(v) = e_{f_k}(v) - \delta, \tag{1.52b}$$

the flow f_{k+1} and excess $e_{f_{k+1}}$ being equal to f_k and e_{f_k}, respectively, on any other arcs and nodes. Given an active node v for which no arc $v \to w$ (for $w \in E(v)$) is admissible, the *relabel* operation updates the distance label of node v such that a push (from v) becomes possible (if there are still residual arcs starting from v), that is the change

$$d_{k+1}(v) = \min_{w\in E_{f_k}(v)} d_k(w) + 1. \tag{1.53}$$

Initialization. — The distance labelling d_0 is initialized for each node v as the distance to the sink node t (that is the least number of nodes on a path from v to t); the source s has by assumption the maximal label n. The initial preflow f_0 is set to zero on E, as well as the excess e_{f_0} on $V \setminus \{s\}$; $e_{f_0}(s)$ is chosen as an upper bound on the potential flow value (for instance, $\sum_{v\to w\in E} u(v \to w)$).

Discharge operation and main loop. — Since it aims to eventually obtain a flow f, the push-relabel method must eliminate the excess at every active node. It therefore successively considers each active node and attempts to *discharge* it from the excess flow, as explained in Algorithm 1.1.

Algorithm 1.1 (Push-relabel: discharge operation)

An active node v is given, as well as an initialized iteration counter k.

Step 1. Select an unconsidered node $w \in E_{f_k}(v)$.

Step 2. If the arc $v \to w$ is admissible, perform a push from v to w, to update f_k to f_{k+1} following (1.52), and increment k by 1.

Step 3. If the node v has become inactive, stop.

Step 4. If there is still unconsidered nodes in $E_{f_k}(v)$, return to Step 1.

Step 5. Perform a relabel of node v to update d_k to d_{k+1} as in (1.53), and increment k by 1.

In Step 1 of this algorithm, the selection of the residual arc $v \to w$ is performed in a fixed but arbitrary order (typically, the input order of the arcs). The push-relabel method then consists in a sequence of discharge operations performed on the remaining active nodes of the network, yielding Algorithm 1.2.

Algorithm 1.2 (Push-relabel method)

A flow network (V, E, s, t, u) is given.

Step 0. Initialization: Set $k = 0$, $f_0 = 0$, $e_{f_0} = 0$,

$$e_{f_0}(s) = \sum_{v \to w \in E} u(v \to w).$$

Compute d_0 as the distance to the sink t, and set $d_0(s) = n$.

Step 1. Node selection: Select an active node v. If none exists, stop.

Step 2. Discharge: Perform the operation $discharge(v)$ and return to Step 1.

Several strategies were designed to decide the order in which the algorithm considers the active nodes in Step 1 (see for instance, Ahuja *et al.*, 1997, and Cherkassky and Goldberg, 1997). In particular, the *lowest-label* strategy orders

the active nodes in increasing order of distance label. In Section 5.4, we choose this strategy in the push-relabel method, and explain our decision on the basis of the flow problem to solve. The previous references nevertheless also report good performance with this strategy on a broader class of flow problems.

Feasible flow problems. — The push-relabel method can be used to solve alternative flow problems, for instance *feasible flow problems*. They consist in identifying a flow f in a given network (V, E, u) satisfying the capacity constraints (1.49), and enforcing the demand at each node, in the sense that

$$e_f(v) = b(v) \tag{1.54}$$

for every node $v \in V$, and given a fixed demand function $b : V \to \mathbb{R}$ satisfying $\sum_{v \in V} b(v) = 0$. This problem may be transformed (see Section 6.2 of Ahuja *et al.*, 1993) in a maximal flow problem by introducing two new nodes: a source node s and a sink node t. For each node v with $b(v) > 0$, an arc $s \to v$ is added with capacity $b(v)$, whereas for each node v with $b(v) < 0$, an arc $v \to t$ is added with capacity $-b(v)$. It can then be shown that the feasible flow problem has a solution if and only if the maximum flow computed on the transformed network saturates all the source and sink arcs.

Multiple sources and sinks. — This method is also applicable in the case of many sources and sinks, since we can augment our network with, on the one hand, a virtual source node linked towards all excess node (the former sources) and on the other hand, a virtual sink node linked from all deficient nodes (the former sinks).

1.5 Performance profiles

When several methods are designed to solve a same class of problems, we would naturally like to decide which one is the best. In optimization, we are in particular interested in methods that are able to solve as many problems as possible (that is *robust* methods), and with the smallest cost possible (that is *efficient* methods). This cost can take several forms, but the more common are the CPU time, the number of iterations and the number of function/gradient/Hessian evaluations. This cost criterion is then used to define the performance of the method on a given problem.

The first element in numerical comparisons of methods is the definition of a set $\mathcal{P}$ of test problems, on which the method performances can be compared. We then run each method on each problem, and measure the performance for each combination, building a table of results. These data are however often hardly readable, in particular when the test problems are numerous. Fortunately, the interpretation of such results became easier with the introduction of performance profiles by Dolan and Moré (2002).

This tool displays a curve for each compared method, which gives the proportion of test problems on which the considered method has a performance within a factor σ of the best. More formally, let $\mathcal{S}$ be the set of compared methods, and $q_{p,s}$ denote the performance of method $s \in \mathcal{S}$ on problem $p \in \mathcal{P}$. We then define the *performance ratio*

$$r_{p,s} \stackrel{\text{def}}{=} \frac{q_{p,s}}{\min\{q_{p,z} : z \in \mathcal{S}\}}, \tag{1.55}$$

which reflects the relative performance of the method s on problem p with respect to the other methods. Note that we set $r_{p,s}$ to infinity when the method s fails to solve the problem p. The performance profile curves are finally given by

$$p_s(\sigma) \stackrel{\text{def}}{=} \frac{|\{p \in \mathcal{P} : r_{p,s} \leqslant \sigma\}|}{|\mathcal{P}|} \qquad \text{for } \sigma \geqslant 1. \tag{1.56}$$

On the one hand, the efficiency of the methods can then be compared by observing the values of $p_s(\sigma)$ for small values of σ (the left part of the graph). In particular $p_s(1)$ gives the proportion of problems on which the method s is the most efficient. On the other hand, robustness may also be studied by looking at the performance profile for large values of σ (the right part of the graph). In particular, $p_s(\infty)$ gives the number of problems that can be solved by method s (irrelative to the potential cost). Visually speaking, the higher a method is displayed on a performance profile, the better it performs.

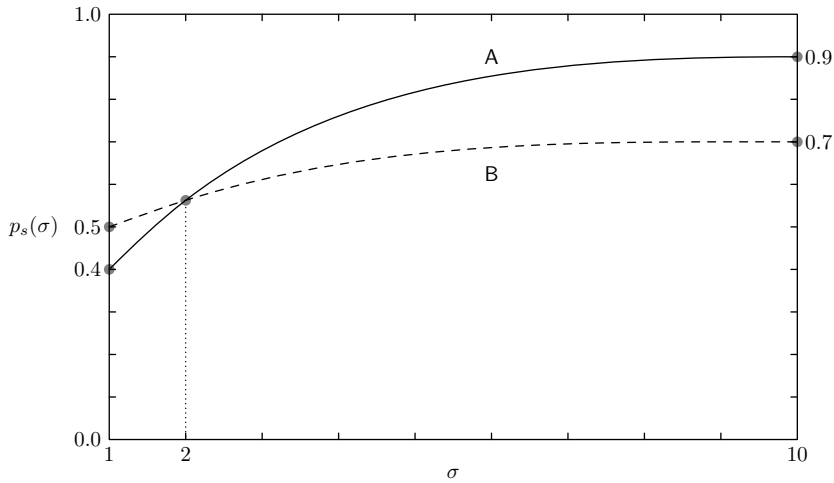


Figure 1.1 — *Example of performance profile.*

Consider for instance the performance profile displayed in Figure 1.1, and assume that we use a time criterion. The performance of an algorithm A is represented by the solid curve, while that of a second algorithm B is represented

by the dashed curve. We observe the method's efficiency on the left side of the graph: the algorithm B is faster than algorithm A on 50 % of the problems, while algorithm A outperforms algorithm B on 40 % of the problems. Note that the sum $\sum_s p_s(1)$ may not always be 1. Problems that are solved with equal speed are indeed counted twice, whereas problems that none manage to solve remain uncounted. On the right side of the profile, we observe that if we let enough time to algorithm A (here, around 10 times that of the best performer between A and B), it solves 90 % of the problems, while algorithm B is only able to solve 70 % of them within the same time scale. Algorithm A is therefore more robust, but less efficient than algorithm B. The central part of the performance profile show the behaviours of the algorithms between these two ends. In this case, we observe that algorithm A becomes better than algorithm B if it is allowed to take twice the time needed by the best performer.

Chapter 2

Fundamentals of nonlinear optimization

While the concept of *optimization* refers to both maximization and minimization problems, only the latter are traditionally considered in the literature, since each class of problems is the other side of the other class. Indeed, given an objective function f,

$$\max f(x) = -\min[-f(x)]$$

and each maximization problem can thus be directly rewritten as its counterpart minimization problem. We therefore only consider minimization problems in what follows.

We first explicit in Section 2.1 what is a solution of a minimization problem, and then present in Section 2.2 necessary and sufficient conditions on the objective function f to obtain such solutions. The next sections are devoted to methods for solving (unconstrained) optimization problems. We start from the fundamental Newton's method in Section 2.3, and then proceed with the derived quasi-Newton methods in Section 2.4. Considering a major drawback of the preceding methods, we present afterwards three classes of methods that address this drawback, namely the linesearch methods in Section 2.5, the conjugate gradient methods in Section 2.6, and the trust-region methods in Section 2.7. We conclude in Section 2.8 with an alternative method from the class of trust-region methods that we developed in collaboration with Fabian Bastin, Mélodie Mouffe, Philippe Toint and Dimitri Tomanos.

When discussing the convergence properties of these methods, we use several constants of the form κ_{xyz}. We refer the reader to the list of notations on page 208 for a quick recall of their meaning.

2.1 Characterization of solutions

Consider the general optimization problem

$$\min_{x \in \mathcal{S}} f(x). \tag{2.1}$$

where f is an objective function which maps $\mathbb{R}^n$ into $\mathbb{R}$, and where the feasible domain $\mathcal{S}$ is included in $\mathbb{R}^n$. We now explicit and characterize what is a solution of the problem (2.1). A point $x_* \in \mathcal{S}$ is a *global minimizer* of f over $\mathcal{S}$ if the value of the objective function f at this point x_* is smaller than its value at any other point x of the feasible domain $\mathcal{S}$, that is

$$f(x_*) \leqslant f(x) \quad \text{for all } x \in \mathcal{S}. \tag{2.2}$$

When the objective function f is nonconvex, finding a global solution of the problem (2.1) can however be a very complicated task. As a consequence, we only often aim at finding a point x_* that minimizes the objective function f only over some region around x_*, and not necessarily over the whole feasible domain $\mathcal{S}$. Such a point $x_* \in \mathcal{S}$ is a (weak) *local minimizer* of f, meaning that there exists a neighbourhood $\mathcal{N}$ of x_* such that

$$f(x_*) \leqslant f(x) \quad \text{for all } x \in \mathcal{N} \cap \mathcal{S}. \tag{2.3}$$

It is moreover a *strict local minimizer* of f if there exists a neighbourhood $\mathcal{N}$ of x_* such that

$$f(x_*) < f(x) \quad \text{for all } x \in (\mathcal{N} \cap \mathcal{S}) \setminus \{x_*\}. \tag{2.4}$$

A local minimizer x_* is *isolated* if there is no other local minimizer within some open neighbourhood of x_*. We call *minimum* the value of the objective function f at a minimizer. Figure 2.1 illustrates the different kinds of minimizers aforementioned.

2.2 Optimality conditions

Optimality conditions are results that allow to identify minimizers more easily than with the conditions (2.2) – (2.4). Let us consider now only the unconstrained minimization problem

$$\min_{x \in \mathbb{R}^n} f(x). \tag{2.5}$$

First, we present two necessary optimality conditions for this problem. These are conditions on the derivatives of the function f that must be satisfied at local minimizers of (2.5).

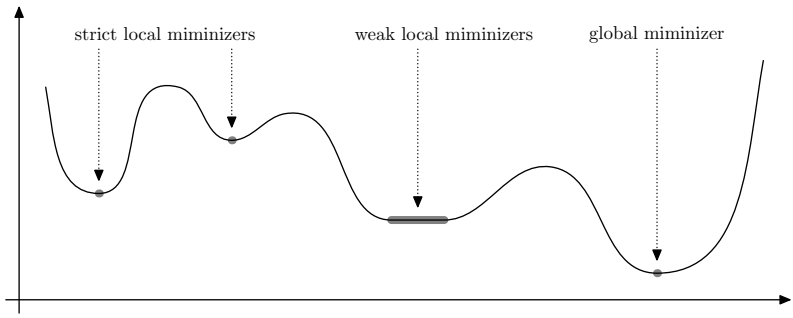


Figure 2.1 — *Examples of local and global minimizers in one dimension.*

Theorem 2.1 (First-order necessary condition) *Suppose that x_* is a local minimizer of problem* (2.5) *and that f is continuously differentiable in an open neighbourhood of x_*. Then*

$$\nabla f(x_*) = 0. \tag{2.6}$$

A point x_* satisfying (2.6) is a *first-order critical point* (or *first-order stationary point*) of f. The first-order necessary condition thus implies that any local minimizer of a continuously differentiable function is a first-order critical point.

Theorem 2.2 (Second-order necessary condition) *Suppose that x_* is a local minimizer of problem* (2.5) *and that f is twice-continuously differentiable in an open neighbourhood of x_*. Then*

$$\nabla f(x_*) = 0 \ \text{ and } \ \nabla^2 f(x_*) \succcurlyeq 0. \tag{2.7}$$

A point x_* satisfying (2.7) is a *second-order critical point* (or *second-order stationary point*) of f. The second-order necessary condition thus implies that any local minimizer of a twice-continuously differentiable function is a second-order critical point.

We now present a sufficient optimality condition. This is a condition on the derivatives of the function f under which a point is a (strict) local minimizer of problem (2.5).

Theorem 2.3 (Second-order sufficient condition) *Suppose that f is twice-continuously differentiable in an open neighbourhood of some point x_* at which*

$$\nabla f(x_*) = 0 \;\; \textit{and} \;\; \nabla^2 f(x_*) \succ 0. \tag{2.8}$$

Then, x_* is a strict, isolated local minimizer of problem (2.5).

If the objective function f is convex, then the optimality conditions are simpler and can be gathered in the following result (see Luenberger, 1969, or Rockafellar, 1970).

Theorem 2.4 (Convex programming) *Suppose that the objective function f is convex on $\mathbb{R}^n$ and continuously differentiable. Then, the point x_* is a global minimizer of problem* (2.5) *if and only if*

$$\nabla f(x_*) = 0. \tag{2.9}$$

Further details on optimality conditions (in particular those for constrained optimization) may be found in Fiacco and McCormick (1968), Mangasarian (1979), Gill, Murray and Wright (1981), Fletcher (1987), Conn, Gould and Toint (2000), and Nocedal and Wright (2006).

2.3 Newton's method

Newton's method is fundamental for the unconstrained optimization of twice differentiable functions. It is an iterative method that generates a sequence of iterates x_k, which is expected to converge to a critical point of (2.5). In what follows, we use the following notations

$$f_k \stackrel{\text{def}}{=} f(x_k) \;\; \text{and} \;\; g_k \stackrel{\text{def}}{=} \nabla f(x_k). \tag{2.10}$$

At each iteration, Newton's method minimizes the quadratic Taylor's model q_k built around the current iterate x_k, that is the function defined by

$$q_k(x_k + s_k) = f_k + \langle g_k, s_k \rangle + \frac{1}{2} \left\langle s_k, \nabla^2 f(x_k) s_k \right\rangle. \tag{2.11}$$

If the Hessian matrix $\nabla^2 f(x_k)$ is positive definite, then the model q_k is strictly convex. The (global) minimizer of (2.11) is therefore given by the unique solution of the so-called *Newton equation*

$$\nabla^2 f(x_k) s_k^{\mathrm{N}} = -g_k, \tag{2.12}$$

because of Theorem 2.4 on convex programming. The resulting vector s_k^{N} is called the *Newton step* or *Newton direction*. The current iterate is then updated using this step:

$$x_{k+1} = x_k + s_k^{\mathrm{N}}, \tag{2.13}$$

and this process is repeated until convergence to a critical point as summarized in Algorithm 2.1.

Algorithm 2.1 (Newton's method)

Given a starting point x_0, set $k = 0$. Then, until convergence:

Step 1. Step computation: Compute the Newton's step s_k^{N} by solving the linear system (2.12).

Step 2. Iterate update: Set $x_{k+1} = x_k + s_k^{\mathrm{N}}$, increment k by 1, and return to Step 1.

Note that the step s_k^{N} may lead to an increase in the objective function value. This is especially the case when the Hessian matrix $\nabla^2 f(x_k)$ is not positive definite, which implies that the objective function is increasing at first-order along that direction. This is the first drawback of the method. However, if we stay sufficiently close to a second-order critical point, the Hessian remains positive definite, yielding steps that appropriately decrease the objective function.

The Hessian computation may be a tough task, yielding the design of the quasi-Newton methods presented in the next section and opening the field of Hessian approximation, which is the main topic of this thesis (see Part II).

Convergence. — Assume that x_* is a second-order critical point and that the Hessian $\nabla^2 f$ is Lipschitz continuous in a neighbourhood of this point. If the starting point x_0 is chosen close enough to x_*, then the sequence of iterates x_k generated by Newton's method converges *quadratically* to x_* (see Ortega and Rheinboldt (1970) for a thorough discussion on the method convergence, or Dennis and Schnabel (1983) for a summary). Newton's method therefore displays a very attractive rate of convergence, but is only *locally* convergent. This lack of global convergence is the second drawback of the method. It will be addressed by the methods presented in Sections 2.5 and following.

2.4 Quasi-Newton methods

Contrary to Newton's method, quasi-Newton methods no longer require the evaluation of the objective function Hessian, but differ from the former method only by the fact that an approximation B_k of the Hessian matrix $\nabla^2 f(x_k)$ is

used instead of the exact Hessian to define the quadratic model to minimize at each iteration; so we now use the model

$$q_k(x_k + s) = f_k + \langle g_k, s\rangle + \tfrac{1}{2}\langle s, B_k s\rangle . \tag{2.14}$$

for some symmetric matrix B_k capturing (partially) the second-order behaviour of the objective function f around x_k. If positive definiteness of the matrix B_k is maintained throughout the iterations, then the *quasi-Newton step* at iteration k is computed as

$$s_k^{\mathrm{QN}} = -B_k^{-1} g_k, \tag{2.15}$$

and the iterate is updated as

$$x_{k+1} = x_k + s_k^{\mathrm{QN}}. \tag{2.16}$$

Algorithm 2.2 states the framework of quasi-Newton methods, where practical implementations of Step 1 are discussed in Part II.

Algorithm 2.2 (Quasi-Newton methods)

Given a starting point x_0, set $k = 0$. Then, until convergence:

Step 1. Model definition: Build a symmetric positive definite matrix B_k, approximating $\nabla^2 f(x_k)$.

Step 2. Step computation: Compute the quasi-Newton step s_k^{QN} by solving the linear system

$$B_k s_k^{\mathrm{QN}} = -g_k. \tag{2.17}$$

Step 3. Iterate update: Set $x_{k+1} = x_k + s_k^{\mathrm{QN}}$, increment k by 1, and return to Step 1.

Convergence. — Let the objective function f be twice-continuously differentiable, and assume that the sequence $\{x_k\}$ generated by Algorithm 2.2 converges to a second-order critical point x_* of f. Then, this sequence converges *superlinearly* if and only if the quasi-Newton steps approximate sufficiently well the Newton's step, in the sense that

$$\lim_{k\to\infty} \frac{\|[B_k - \nabla^2 f(x_k)]s_k^{\mathrm{QN}}\|}{\|s_k^{\mathrm{QN}}\|} = 0 \tag{2.18}$$

(see Theorem 3.7 of Nocedal and Wright, 2006).

As such, quasi-Newton methods still lack global convergence, which is a problem addressed in the next sections.

2.5 Linesearch methods

Linesearch (see already Cauchy, 1847) is the traditional way to reach global convergence in iterative processes like the (quasi-)Newton methods. At each iteration k, a *search direction* d_k is first built, and then the method decides how far to move along this direction. The next iterate is thus

$$x_{k+1} = x_k + \alpha_k d_k \tag{2.19}$$

for some positive *step length* α_k.

Search direction. — In most cases, linesearch methods require the search direction d_k to be a *descent direction*, that is one such that

$$\langle g_k, d_k \rangle < 0, \tag{2.20}$$

for it ensures that the objective function f can be reduced along this direction. The simplest choice of descent direction is probably $-g_k$, yielding the *steepest descent* method, which converges from any starting point (see Curry, 1944), but unfortunately only linearly. This convergence may be very slow as soon as the problem is slightly poorly scaled. By contrast, Newton's method is independent to change of scales. Better choices include the Newton's direction s_k^{N} and any quasi-Newton direction $B_k^{-1} g_k$ provided that $\nabla^2 f(x_k)$ and B_k, respectively, are positive definite.

Step length. — The ideal step length would be the exact minimizer of the function along the search direction, that is

$$\alpha_k = \operatorname*{arg\,min}_{\alpha > 0} f(x_k + \alpha d_k). \tag{2.21}$$

However, even solving this unidimensional optimization problem can be computationally expensive, mainly in terms of function and gradient evaluations. Besides, it may not be useful far from convergence. An inexact solution α_k fulfilling certain conditions is thus often preferred. In that case, an *inexact linesearch* is said to be performed, by opposition to an *exact linesearch*, in which the exact minimizer (2.21) is used. Let us denote by ϕ the restriction of the function f along the direction d_k, that is the function

$$\phi : \mathbb{R}^+ \to \mathbb{R} : \alpha \mapsto f(x_k + \alpha d_k), \tag{2.22}$$

whose derivative is $\phi' : \alpha \mapsto \langle \nabla f(x_k + \alpha d_k), d_k \rangle$. Based on work by Armijo (1966) and Goldstein (1967), the *Wolfe conditions* are the most commonly required:

$$\phi(\alpha_k) \leqslant \phi(0) + \delta \alpha_k \phi'(0), \tag{2.23a}$$

$$\phi'(\alpha_k) \geqslant \sigma \phi'(0), \tag{2.23b}$$

for some parameters $0 < \delta < \sigma < 1$. The equation (2.23a) is called the *Armijo condition* and ensures a *sufficient decrease* in the objective function f. It indeed means that the function reduction obtained at $x_k + \alpha_k d_k$ is at least a fraction (δ) of that achievable by the linear approximation of f at x_k. On the other hand, the equation (2.23b), usually called the *curvature condition*, prevents unacceptably short step lengths. Wolfe conditions can yet be fulfilled by a step length α_k that is far from a minimizer of the function f along the direction d_k. This drawback may be addressed by slightly modifying the curvature condition (2.23b), yielding the *strong Wolfe conditions*:

$$\phi(\alpha_k) \leqslant \phi(0) + \delta\alpha_k\phi'(0), \tag{2.24a}$$

$$|\phi'(\alpha_k)| \geqslant \sigma\,|\phi'(0)|\,. \tag{2.24b}$$

Wolfe (1969, 1971) shows that there always exists an open interval of step lengths α_k satisfying (2.23), provided that the function f is continuously differentiable and bounded below along the descent direction d_k. The same result holds for the strong Wolfe conditions (2.24). Dennis and Moré (1974) show that a linesearch method using quasi-Newton descent directions (2.15) satisfying (2.18) and ensuring the Wolfe conditions still converges superlinearly.

Recently, Hager and Zhang (2005) introduce a new set of conditions for their linesearch method, called *approximate Wolfe conditions*:

$$(2\delta - 1)\phi'(0) \geqslant \phi'(\alpha_k) \geqslant \sigma\phi'(0), \tag{2.25}$$

where $0 < \delta < \frac{1}{2}$ and $\delta \leqslant \sigma < 1$. While the second inequality in (2.25) is the same as the Wolfe curvature condition (2.23b), the first one is an approximation of the Armijo condition (2.23a) to the order α_k^2. This modification intends to obtain a better accuracy, since the function derivatives, in finite precision arithmetic, should be more reliable than the function value near local minimizers.

We now describe two algorithms for step lengths computation, namely that of Dennis and Schnabel in Section 2.5.1 and that of Hager and Zhang in Section 2.5.2. We also refer the reader to the well-known linesearch method of Moré and Thuente (1994), and to Chapter 3 of Nocedal and Wright (2006) for further information on this class of methods.

2.5.1 Dennis-Schnabel linesearch method

The Dennis-Schnabel linesearch (see Algorithm 6.3.1mod of Dennis and Schnabel, 1983) determines positive step lengths that ensure the satisfaction of the Wolfe conditions (2.23). Given a descent direction d_k, their algorithm computes iteratively the step length α_k starting from the unit value, and stops as soon as a satisfying step length is found. In the next description, we denote by α_k^ℓ the the step length at the ℓ-th inner iteration of the method. The latter distinguishes three cases at every inner iteration ℓ. First, if only the Armijo

condition is satisfied and if $\alpha_k^\ell \geqslant 1$, the step length is doubled. Second, if the Armijo condition is not satisfied and if $\alpha_k^\ell \leqslant 1$, then a backtracking step is performed to obtain a step length $\alpha_k^{\ell+1}$ smaller than α_k^ℓ. More precisely, a cubic unidimensional model of the function f along the direction d_k is built by interpolating $\phi(0)$, $\phi'(0)$, $\phi(\alpha_k^\ell)$, and $\phi(\alpha_k^{\ell-1})$ (as soon as $\ell > 1$; otherwise the model is only quadratic), and the minimizer of this model is then chosen as the new step length $\alpha_k^{\ell+1}$. Otherwise, a sequence of quadratic interpolation steps (using the information $\phi(\alpha_k^\ell)$, $\phi(\alpha_k^{\ell-1})$ and $\phi'(\alpha_k^\ell)$) is performed until the Wolfe conditions are satisfied.

Algorithm 2.3 (Dennis-Schnabel: step length selection)

Two constants δ and σ are given and satisfy

$$0 < \delta < \sigma < 1. \tag{2.26}$$

Step 0. Initialization: Set $\alpha_k^0 = 1$, and $\ell = 0$.

Step 1. Stopping criterion: If $\phi(\alpha_k^\ell) \leqslant \phi(0) + \delta\alpha_k^\ell\phi'(0)$ and $\phi'(\alpha_k^\ell) \geqslant \sigma\phi'(0)$, then stop.

Step 2. Doubling step: If $\phi(\alpha_k^\ell) \leqslant \phi(0) + \delta\alpha_k^\ell\phi'(0)$, $\phi'(\alpha_k^\ell) < \sigma\phi'(0)$ and $\alpha_k^\ell \geqslant 1$, then set $\alpha_k^{\ell+1} = 2\alpha_k^\ell$, increment ℓ by 1, and return to Step 1.

Step 3. Backtracking step: If $\phi(\alpha_k^\ell) > \phi(0) + \delta\alpha_k^\ell\phi'(0)$ and $\alpha_k^\ell \leqslant 1$, then set $\alpha_k^{\ell+1}$ to the minimizer of the cubic model interpolating $\phi(0)$, $\phi'(0)$, $\phi(\alpha_k^\ell)$, and, as soon as $\ell > 1$, $\phi(\alpha_k^{\ell-1})$; increment ℓ by 1, and return to Step 1.

Step 4. Interpolation steps: While $\phi(\alpha_k^\ell) > \phi(0) + \delta\alpha_k^\ell\phi'(0)$ or $\phi'(\alpha_k^\ell) < \sigma\phi'(0)$, set $\alpha_k^{\ell+1}$ to the minimizer to the quadratic model interpolating $\phi(\alpha_k)$, $\phi(\alpha_k^{\ell-1})$ and $\phi'(\alpha_k^\ell)$, and increment ℓ by 1.

The recommended parameters for the Wolfe conditions herein are $\delta = 10^{-4}$ and $\sigma = 0.9$.

2.5.2 Hager-Zhang linesearch method

We here consider a particular outer iteration k of the linesearch method, and explicit how Hager and Zhang (2005) (see also Hager and Zhang, 2006*a*, for further numerical considerations) determine the step length α_k.

The Hager-Zhang linesearch method builds a sequence of nested *bracketing intervals* $[\mu_\ell, \nu_\ell]$, which "converges" to a satisfying step length α_k. These

intervals satisfy the *opposite slope condition*:

$$\phi(\mu_\ell) \leqslant \phi(0) + \epsilon_k, \quad \phi'(\mu_\ell) < 0 \quad \text{and} \quad \phi'(\nu_\ell) \geqslant 0. \tag{2.27}$$

Note that the focus for accuracy in finite precision arithmetic again yields conditions involving mostly the function derivatives, instead of function values (like with the linesearch of Moré and Thuente, 1994).

We first describe the termination condition of this construction, then explain how the bracketing interval sequence is initialized, and afterwards how it is iteratively built; finally, we state the complete Hager-Zhang algorithm.

Termination condition. — Their linesearch method may ensure the satisfaction either of the Wolfe conditions (2.23), or of the approximate Wolfe conditions (2.25) combined to the additional reduction condition

$$\phi(\alpha_k) \leqslant \phi(0) + \epsilon_k \tag{2.28}$$

where $\epsilon_k \geqslant 0$ is an estimate for the error in the evaluation of f at x_k, for instance $\epsilon\,|f_k|$ for some small constant $\epsilon \geqslant 0$. In practice, since numerical errors may occur when consecutive iterates become close, the method switches from the former set of conditions to the latter as soon as

$$|f_{k+1} - f_k| \leqslant \omega C_k, \tag{2.29}$$

where

$$Q_k = 1 + Q_{k-1}\Upsilon, \qquad Q_{-1} = 0, \tag{2.30a}$$
$$C_k = C_{k-1} + (|f_k| - C_{k-1})/Q_k, \qquad C_{-1} = 0, \tag{2.30b}$$

for some parameters Υ and ω in the interval $[0, 1]$. Note that C_k is a weighted mean of the values $|f_k|$, where Υ is a decay factor for the weights corresponding to past iterations; hence, $C_k = |f_k|$ when $\Upsilon = 0$, while C_k is the arithmetic mean of the values $|f_k|$ when $\Upsilon = 1$. Hager and Zhang (2006*a*) suggest to choose $\delta = 0.1$, $\sigma = 0.9$ and $\epsilon = 10^{-6}$ in the (approximate) Wolfe conditions, and $\Upsilon = 0.7$ and $\omega = 10^{-3}$ in the switch (2.29) – (2.30).

Bracketing interval initialization. — We first need to build an initial bracketing interval $[\mu_0, \nu_0]$ that satisfies the opposite slope condition (2.27). Given a first step length guess α^0 (determined on the basis of the previous step length α_{k-1}, see Hager and Zhang, 2006*a*), Algorithm 2.4 determines such an interval.

Algorithm 2.4 (Hager-Zhang: bracketing interval initialization)

An initial step length guess α^0 and two constants $\theta \in (0, 1)$ and $\rho > 1$ are given.

Step 0. Set $\varsigma_{-1} = 0$ and $\varsigma_0 = \alpha^0$. Initialize the counter j to 0.

Step 1. If $\phi'(\varsigma_j) \geqslant 0$, then terminate with $[\mu_0, \nu_0] = [\varsigma_i, \varsigma_j]$ where

$$i = \max\{i < j : \phi(\varsigma_i) \leqslant \phi(0) + \epsilon_k\}. \tag{2.31}$$

Step 2. If $\phi(\varsigma_j) \leqslant \phi(0) + \epsilon_k$, then set $\varsigma_{j+1} = \rho\varsigma_j$, increment j by 1, and return to Step 1.

Step 3. Initialize the interval $[\upsilon_0^L, \upsilon_0^R] = [0, \varsigma_j]$, and the counter i to 0. Compute the convex combination $\upsilon_0 = (1-\theta)\upsilon_0^L + \theta\upsilon_0^R$.

Step 4. While $\phi'(\upsilon_i) < 0$, update the interval

$$[\upsilon_{i+1}^L, \upsilon_{i+1}^R] = \begin{cases} [\upsilon_i^L, \upsilon_i] & \text{if } \phi(\upsilon_i) > \phi(0) + \epsilon_k, \\ [\upsilon_i, \upsilon_i^R] & \text{if } \phi(\upsilon_i) \leqslant \phi(0) + \epsilon_k, \end{cases} \tag{2.32}$$

and the convex combination $\upsilon_{i+1} = (1-\theta)\upsilon_{i+1}^L + \theta\upsilon_{i+1}^R$, and increment i by 1.

Step 5. Terminate with the interval $[\mu_0, \nu_0] = [\upsilon_{i+1}^L, \upsilon_{i+1}]$.

Algorithm 2.4 needs a few comments. Firstly, in Step 1, the value ς_i is well-defined, since $i = -1$ always fulfils the conditions $i < j$ and $\phi(\varsigma_i) \leqslant \phi(0) + \epsilon_k$. Moreover, we have $\phi'(\varsigma_i) < 0$, since j is the first index for which $\phi'(\varsigma_j) \geqslant 0$. Secondly, in Step 4, a sequence of nested intervals $[\upsilon_i^L, \upsilon_i^R]$ is created such that

$$\phi'(\upsilon_i^L) < 0, \quad \phi'(\upsilon_i^R) < 0 \quad \text{and} \quad \phi(\upsilon_i^L) \leqslant \phi(0) + \epsilon_k < \phi(\upsilon_i^R). \tag{2.33}$$

This process ends in finitely many iterations, since the length of the interval $\upsilon_i^R - \upsilon_i^L$ tends to 0, while there always exists an interior point at which $\phi' > 0$ because of the mean-value theorem with $\phi(\upsilon_i^L) < \phi(\upsilon_i^R)$. Hager and Zhang (2006*a*) suggest to choose $\theta = \frac{1}{2}$ (which corresponds to a bisection method) and to set the expansion factor ρ to 5.

Bracketing interval update. — We now describe the bracketing interval updating process. It consists of two stages: a *double secant step* and possibly, an additional *interval update*. We first define the secant step operation as

$$\operatorname{secant}(a, b) = \frac{a\phi'(b) - b\phi'(a)}{\phi'(b) - \phi'(a)}. \tag{2.34}$$

Note that this operation is symmetric and that, given an interval $[a, b]$ satisfying the opposite slope condition, the new point $\operatorname{secant}(a, b)$ is the barycentre of

points a and b with weights $|\phi'(b)|$ and $|\phi'(a)|$, respectively; hence secant$(a, b) \in [a, b]$.

Given a bracketing interval $[\mu, \nu]$ satisfying the opposite slope condition, and an update point ς, the *interval updating* procedure described in Algorithm 2.5 returns a nested interval $[\mu^+, \nu^+]$, which also satisfies the opposite slope condition. This operation is noted

$$[\mu^+, \nu^+] = \text{update}(\mu, \nu, \varsigma). \tag{2.35}$$

Algorithm 2.5 (Hager-Zhang: interval updating)

An interval $[\mu, \nu]$ satisfying (2.27), an update point ς, and a constant $\theta \in (0, 1)$ are given.

Step 0. If $\varsigma \notin (a, b)$, then terminate with $[\mu^+, \nu^+] = [\mu, \nu]$.

Step 1. If $\phi'(\varsigma) \geqslant 0$, then terminate with $[\mu^+, \nu^+] = [\mu, \varsigma]$.

Step 2. If $\phi(\varsigma) \leqslant \phi(0) + \epsilon_k$, then terminate with $[\mu^+, \nu^+] = [\varsigma, \nu]$.

Step 3. Initialize the interval $[v_0^L, v_0^R] = [\mu, \varsigma]$, and the counter i to 0. Compute the convex combination $v_0 = (1 - \theta)v_0^L + \theta v_0^R$.

Step 4. While $\phi'(v_i) < 0$, update the interval

$$[v_{i+1}^L, v_{i+1}^R] = \begin{cases} [v_i^L, v_i] & \text{if } \phi(v_i) > \phi(0) + \epsilon_k, \\ [v_i, v_i^R] & \text{if } \phi(v_i) \leqslant \phi(0) + \epsilon_k, \end{cases} \tag{2.36}$$

and the convex combination $v_{i+1} = (1 - \theta)v_{i+1}^L + \theta v_{i+1}^R$, and increment i by 1.

Step 5. Terminate with the interval $[\mu^+, \nu^+] = [v_{i+1}^L, v_{i+1}]$.

Note that Steps 1 and 2 perform a straightforward replacement of a bound of the interval $[\mu, \nu]$ by the update point ς whenever it is possible. Otherwise, the same strategy as in the initialization phase is used to produce a nested interval satisfying the opposite slope condition, starting from the interval $[\mu, \varsigma]$. This latter case is the only one in which both bounds of the interval may be changed.

The double secant step described Algorithm 2.6 updates the interval $[\mu, \nu]$ (once or twice) on the basis of a secant step, yielding a nested interval $[\mu^+, \nu^+]$, which also satisfies the opposite slope condition. This operation, noted

$$[\mu^+, \nu^+] = \text{secant}^2(\mu, \nu), \tag{2.37}$$

intends to change both bounds of the interval such that $\mu < \mu^+ < \nu^+ < \nu$. Therefore, if both bounds of the interval $[\mu, \nu]$ are changed in a single interval

update operation (that is the case of Step 4 in the algorithm), no second interval update is performed.

Algorithm 2.6 (Hager-Zhang: double secant step)

An interval $[\mu, \nu]$ satisfying (2.27) is given.

Step 1. Compute $\varsigma = \text{secant}(\mu, \nu)$ and $[\hat{\mu}, \hat{\nu}] = \text{update}(\mu, \nu, \varsigma)$.

Step 2. If $\hat{\mu} = \varsigma$, then terminate with

$$[\mu^+, \nu^+] = \text{update}(\hat{\mu}, \hat{\nu}, \text{secant}(\mu, \hat{\mu})). \tag{2.38}$$

Step 3. If $\hat{\nu} = \varsigma$, then terminate with

$$[\mu^+, \nu^+] = \text{update}(\hat{\mu}, \hat{\nu}, \text{secant}(\nu, \hat{\nu})). \tag{2.39}$$

Step 4. Otherwise, terminate with $[\mu^+, \nu^+] = [\hat{\mu}, \hat{\nu}]$.

Complete algorithm. — We now finally present in Algorithm 2.7 the Hager-Zhang linesearch method for computing the step length α_k. Note that the algorithm stops as soon as a step length α_k satisfying the termination conditions is generated at any stage of the procedure. Each time a new point is considered in the process (including the called algorithms for initial bracketing interval, double secant step, and interval updating), the termination conditions are checked for this step length. Observe also that new values of the step length are always generated one by one, so this step length selection is well defined. Indeed, when an interval is updated, either one of its bounds has already been tested and considered as an inappropriate step length, or these bounds have been built in a well-defined order.

Algorithm 2.7 (Hager-Zhang: step length selection)

The previous step length α_{k-1} (if $k > 0$) and a constant $\gamma \in (0, 1)$ are given. The algorithm stops as soon as a point satisfying the termination conditions is generated.

Step 0. Determine an initial bracketing interval $[\mu_0, \nu_0]$ using Algorithm 2.4, and set the iteration counter ℓ to 0.

Step 1. Using Algorithm 2.6, perform a double secant step:

$$[\mu^+, \nu^+] = \text{secant}^2(\mu_\ell, \nu_\ell), \tag{2.40}$$

Step 2. If $(\mu^+ - \nu^+) > \gamma(\mu_\ell - \nu_\ell)$, then perform an additional interval updating:

$$[\mu_{\ell+1}, \nu_{\ell+1}] = \text{update}\left(\mu^+, \nu^+, \tfrac{\mu^+ + \nu^+}{2}\right), \tag{2.41}$$

using Algorithm 2.5; otherwise, set $[\mu_{\ell+1}, \nu_{\ell+1}] = [\mu^+, \nu^+]$. Increment ℓ by 1, and return to Step 1

The suggested value for parameter γ in Step 2 is 0.66. This choice ensures that the bracketing interval length decreases by at least a factor 2/3 at each iteration.

2.6 Conjugate gradient methods

Let us first introduce the concept of *conjugacy*. Given a symmetric positive definite matrix $A \in \mathbb{R}^{n\times n}$, two nonzero directions d_1 and d_2 of $\mathbb{R}^n$ are *A-conjugate* if

$$\langle d_1, Ad_2\rangle = 0. \tag{2.42}$$

If no confusion exists about the matrix with respect to which conjugacy occurs, we simply say that these directions are conjugate.

We first explain in Section 2.6.1 the original conjugate gradient method for positive definite linear systems. In Section 2.6.2, we then present a variation of this method to solve quadratic problems with ball constraint. Finally, Section 2.6.3 is devoted to the more general case of nonlinear function minimization.

2.6.1 Linear conjugate gradient method

Introduced by Hestenes and Stiefel (1952), the (linear) *conjugate gradient* method aims to solve iteratively linear systems of the form

$$Ax = b \tag{2.43}$$

where the matrix $A \in \mathbb{R}^{n\times n}$ is symmetric positive definite, and where b and x are vectors of $\mathbb{R}^n$. Due to Theorem 2.4 on convex programming, this problem (2.43) is equivalent to the minimization of the convex quadratic function

$$q : x \mapsto \frac{1}{2}\langle x, Ax\rangle - \langle b, x\rangle, \tag{2.44}$$

its gradient being then $\nabla q : x \mapsto Ax - b$.

The main idea of the conjugate gradient method is to minimize the objective function in nested subspaces of increasing dimension. Indeed, at each iteration k, the algorithm builds a new iterate x_{k+1} that minimizes the function q over the subspace $x_0 + \mathcal{K}(A, b, k)$ where

$$\mathcal{K}(A, b, k) \stackrel{\text{def}}{=} \text{span}\left\{b, Ab, A^2b, \ldots, A^{k-1}b\right\} \tag{2.45}$$

is called a *Krylov subspace*. These successive minimization steps may fortunately be performed in an efficient way, since the next iterate provably stays in a direction conjugate to those spanning $\mathcal{K}(A, b, k)$. The conjugate gradient method therefore consists of two main parts: the building of a set of conjugate directions d_k and the (unidimensional) minimizations along these directions. Interestingly, the conjugate gradient method only uses the former direction d_{k-1} to compute the conjugate direction d_k:

$$d_{k+1} = -g_{k+1} + \beta_k d_k \;\text{ with }\; \beta_k = \frac{\langle g_{k+1}, g_{k+1}\rangle}{\langle g_k, g_k\rangle}, \tag{2.46}$$

where $g_k = \nabla q(x_k) = Ax_k - b$ and $d_0 = -g_0$. Hence the name of the method, which uses conjugate directions starting from the gradient direction. On the other hand, as the function q is quadratic, minimizing $q(x_k + \alpha d_k)$ in the variable α easily yields the solution

$$\alpha_k = \frac{\langle g_k, g_k\rangle}{\langle d_k, Ad_k\rangle}. \tag{2.47}$$

Before stating more formally the conjugate gradient algorithm, we briefly discuss its convergence, and present a common modification to improve its performance.

Convergence. — The convergence of the conjugate gradient method is mainly determined by the spectrum of the matrix A. In exact arithmetic, the solution is found in at most as many iterations as the number of distinct eigenvalues of the matrix A (which is n in the worst case). The convergence speed is linear and depends on the condition number of the matrix A. In finite precision arithmetic, the cumulative loss of conjugacy when building the set of directions d_k may however significantly increase that number of iterations, and prevent to obtain an approximate solution with arbitrary small error (see the concept of *maximal attainable accuracy* in Meurant, 2006).

Preconditioning. — To accelerate the method convergence, *preconditioning* is often applied, that is an appropriate change of variables from x to $\hat{x}$, through a nonsingular matrix S, given by $\hat{x} = Sx$. Hence the linear system (2.43) is equivalent to

$$(S^{-T}AS^{-1})\hat{x} = S^{-T}b, \tag{2.48}$$

but the matrix $S^{-T}AS^{-1}$ may present a spectrum better suited to a fast convergence of the conjugate gradient method. The preconditioning technique may quite easily be implicitly integrated in the basic method, yielding the following changes:

$$d_{k+1} = -M^{-1}g_{k+1} + \beta_k d_k \quad \text{with} \quad \beta_k = \frac{\langle g_{k+1}, M^{-1}g_{k+1}\rangle}{\langle g_k, M^{-1}g_k\rangle}, \tag{2.49}$$

where $d_0 = -M^{-1}g_0$, and

$$\alpha_k = \frac{\langle g_k, M^{-1}g_k\rangle}{\langle d_k, Ad_k\rangle}, \tag{2.50}$$

where the *preconditioner* matrix $M = S^T S$ is symmetric positive definite by construction. We now summarize all these elements of the preconditioned conjugate gradient method in Algorithm 2.8.

Algorithm 2.8 (Preconditioned Conjugate Gradient method)

Given an initial point x_0 and a symmetric positive definite preconditioner M, initialize $g_0 = Ax_0 - b$, $v_0 = M^{-1}g_0$, $d_0 = -v_0$. Then, for $k = 0, 1, \ldots$ until convergence:

$$\alpha_k = \tfrac{\langle g_k, v_k\rangle}{\langle d_k, Ad_k\rangle} \tag{2.51a}$$
$$x_{k+1} = x_k + \alpha_k d_k \tag{2.51b}$$
$$g_{k+1} = g_k + \alpha_k Ad_k \tag{2.51c}$$
$$v_{k+1} = M^{-1}g_{k+1} \tag{2.51d}$$
$$\beta_k = \tfrac{\langle g_{k+1}, v_{k+1}\rangle}{\langle g_k, v_k\rangle} \tag{2.51e}$$
$$d_{k+1} = -v_{k+1} + \beta_k d_k \tag{2.51f}$$

Numerical cost. — The linear conjugate gradient method is particularly well-suited to sparse large-scale problems because of its small memory requirements (only four vectors). Indeed, the matrix A needs not to be stored explicitly as a matrix; only matrix-vector products by this matrix are required, which are cheap operations when the matrix A is sparse for instance. The preconditioner matrix M should always be chosen such that the preconditioning operation (2.51d) is also cheap.

A deeper coverage of the topic may be found in Sections 5.1 of both Conn *et al.* (2000) and Nocedal and Wright (2006). We also recommend the book of Meurant (2006), which discusses many theoretical and practical aspects of the method, and in particular its behaviour in finite-precision arithmetic.

2.6.2 Truncated conjugate gradient method

The truncated conjugate gradient method was first introduced by Toint (1981*b*) and completed by Steihaug (1983) to solve trust-region subproblems (see Section 2.7 below) in the variable s of the form

$$\min_{\|s\|_M \leqslant \Delta} \langle g, s\rangle + \frac{1}{2}\langle s, Bs\rangle\,, \tag{2.52}$$

where B is a symmetric matrix of $\mathbb{R}^{n\times n}$, and M is a symmetric positive definite matrix of $\mathbb{R}^{n\times n}$. This problem is quite similar to the minimization of the function q given by (2.44), except for the potential non positive definiteness of the matrix B_k and for the ball constraint $\|s\|_M \leqslant \Delta$. Although the truncated conjugate gradient method starts in the same way as the linear conjugate gradient method (with initial point $s_0 = 0$), their ends differ because the former needs to address the two changes in the problem definition. First, if a direction of negative curvature is met, meaning that $\langle d_k, Ad_k\rangle$ is negative, then the quadratic function value could be decreased as much as wanted by sufficiently moving away from the current iterate; so the method follows this direction until it reaches the ball boundary, and then stops. Second, if the conjugate gradient method generates an iterate lying outside the feasible ball, then we turn back (along the conjugate direction) to the point where the ball boundary has been crossed, and also stop. This strategy makes sense, since the norm of the vector s provably increases monotonically along the conjugate gradient iterations; any further iterate would therefore also lie outside the feasible ball.

We state formally the truncated conjugate gradient method in Algorithm 2.9 (still using a preconditioner M as in the previous section), and refer the interested reader to Section 7.5.1 of Conn *et al.* (2000) for more information.

Algorithm 2.9 (Truncated Conjugate gradient method: TCG)

Given a symmetric positive definite preconditioner M, initialize $s_0 = 0$, $g_0 = g$, $v_0 = M^{-1}g_0$, $d_0 = -v_0$ and $k = 0$. Then, until convergence:

Step 1. Curvature condition: Set $\kappa_k = \langle d_k, Bd_k\rangle$. If $\kappa_k \leqslant 0$, go to Step 4.

Step 2. Boundary condition: Set $\alpha_k = \langle g_k, v_k\rangle / \kappa_k$. If $\|s_k + \alpha_k d_k\|_M \geqslant \Delta$, go to Step 4.

Step 3. Conjugate gradient iteration:

$$s_{k+1} = s_k + \alpha_k d_k, \tag{2.53a}$$
$$g_{k+1} = g_k + \alpha_k B d_k, \tag{2.53b}$$
$$v_{k+1} = M^{-1} g_{k+1}, \tag{2.53c}$$
$$\beta_k = \frac{\langle g_{k+1}, v_{k+1} \rangle}{\langle g_k, v_k \rangle}, \tag{2.53d}$$
$$d_{k+1} = -v_{k+1} + \beta_k d_k. \tag{2.53e}$$

Increment k by 1, and return to Step 1.

Step 4. Termination on the boundary: Compute α_k as the positive root of $\|s_k + \alpha_k d_k\|_M = \Delta$, set $s_{k+1} = s_k + \alpha_k d_k$, and stop.

2.6.3 Nonlinear conjugate gradient methods

As their name suggests it, nonlinear conjugate gradient methods aim to solve not only linear systems, but any nonlinear unconstrained minimization problem of the form (2.5). Introduced by Fletcher and Reeves (1964), they are a particular subclass of linesearch methods, since they iteratively minimize the objective function f along some directions. The build of these directions is inspired by that in the linear case, since the search direction is, in both cases, updated from the previous direction following the formula (2.51f), and that the nonlinear methods (with an exact linesearch) applied to the quadratic function (2.44) are furthermore just the linear method. The linear and nonlinear conjugate gradient methods therefore differ by only two elements. First, the linesearch to determine the step length α_k is rarely exact in the nonlinear case, because of its potentially unaffordable cost. Second, there are many variants for the definition of the conjugacy factor β_k in the nonlinear case that still give the same value in the linear case. As we have already discussed the linesearch topic above, we now only focus on the conjugacy factor presentation.

Conjugacy factor. — The first proposition of conjugacy factor appears on the original paper on the linear conjugate gradient method by Hestenes and Stiefel (1952):

$$\beta_k^{\text{HS}} = \frac{\langle g_{k+1}, y_k \rangle}{\langle d_k, y_k \rangle}, \tag{2.54}$$

where $y_k \overset{\text{def}}{=} g_{k+1} - g_k$.

Fletcher and Reeves (1964) yet propose the alternative formula:

$$\beta_k^{\mathrm{FR}} = \frac{\langle g_{k+1}, g_{k+1} \rangle}{\langle g_k, g_k \rangle}, \tag{2.55}$$

which is also most often used in the linear case. If the linesearch fulfils the strong Wolfe conditions with $\sigma < \frac{1}{2}$, then the search directions generated with the Fletcher-Reeves conjugacy factor are guaranteed to be of descent. Al-Baali (1985) shows moreover the global convergence of this method provided that the level set $\mathcal{L} = \{x \in \mathbb{R}^n : f(x) \leqslant f(x_0)\}$ is bounded and that the function f is Lipschitz continuously differentiable in some open neighbourhood of $\mathcal{L}$.

The other classical choice is due to Polak and Ribière (1969) as well as Polyak (1969):

$$\beta_k^{\mathrm{PRP}} = \frac{\langle g_{k+1}, y_k \rangle}{\langle g_k, g_k \rangle}. \tag{2.56}$$

Even the strong Wolfe conditions can not ensure that the search directions generated using β_k^{PRP} are of descent. Moreover, whereas the Polak-Ribière-Polyak method performs better in practise than the Fletcher-Reeves', it may not converge from any starting point even if the linesearch is exact (see Powell, 1984). These drawbacks may yet be addressed by slightly changing the definition of the conjugacy factor:

$$\beta_k^{\mathrm{PRP+}} = \max(0, \beta_k^{\mathrm{PRP}}). \tag{2.57}$$

A practical implementation (called PRP+) of the nonlinear conjugate gradient method using $\beta_k^{\mathrm{PRP+}}$ is co-authored by Guanghui Liu, Jorge Nocedal, Richard Waltz.

More recently, Dai and Yuan (1999) suggest the definition

$$\beta_k^{\mathrm{DY}} = \frac{\langle g_{k+1}, g_{k+1} \rangle}{\langle d_k, y_k \rangle}, \tag{2.58}$$

and, in 2001, recommend its hybridization with the Hestenes-Stiefel formula:

$$\beta_k^{\mathrm{DYHS}} = \max\left[0, \min\left[\beta_k^{\mathrm{DY}}, \beta_k^{\mathrm{HS}}\right]\right]. \tag{2.59}$$

On the other hand, Hager and Zhang (2005) propose the conjugacy factor

$$\beta_k^{\mathrm{HZ}} = \left(y_k - 2d_k \frac{\langle y_k, y_k \rangle}{\langle d_k, y_k \rangle}\right)^T \frac{g_{k+1}}{\langle d_k, y_k \rangle}, \tag{2.60}$$

The methods based on the choices (2.58) – (2.60) generate descent directions provided that their linesearch fulfils the Wolfe conditions. Moreover, Hager and Zhang (2006*a*) present excellent numerical results with these conjugacy factors, the best choice being β_k^{HZ}, and then β_k^{DYHS}.

Preconditioning. — Just as in the linear case, preconditioning may improve the performance of the nonlinear conjugate gradient methods. A classical strategy consists in choosing a variable preconditioner M_k equal to a quasi-Newton approximation of $\nabla^2 f(x_*)^{-1}$.

Algorithm 2.10 (Nonlinear conjugate gradient method: NCG)

Given an initial point x_0 and a symmetric positive definite preconditioner M, initialize $g_0 = \nabla f(x_0)$, $v_0 = M^{-1}g_0$, $d_0 = -v_0$ and $k = 0$. Then, until convergence:

Step 1. Step length computation: Determine a step length α_k by performing a linesearch that minimizes f in direction d_k.

Step 2. Iterate update: Set $x_{k+1} = x_k + \alpha_k d_k$ and compute $g_{k+1} = \nabla f(x_{k+1})$ and $v_{k+1} = M^{-1}g_{k+1}$.

Step 3. Search direction selection: Compute β_k according to the chosen formula, and set $d_{k+1} = -v_{k+1} + \beta_k d_k$. Then increment k by 1, and return to Step 1.

We suggest the interested reader to consult the survey on the topic by Hager and Zhang (2006*b*), or the book by Pytlak (2009) for an extensive coverage.

2.7 Trust-region methods

The concept of trust region was first sketched by Levenberg (1944) to solve nonlinear least-squares problems. Further (independent) developments were then brought by Morrison (1960), Griffith and Stewart (1961), Marquardt (1963) and Goldfeldt, Quandt and Trotter (1966). Finally, Powell (1970) formalized and advocated them as a technique to ensure global convergence of methods in unconstrained optimization. We refer to Section 1.2 of Conn *et al.* (2000) for a complete history of this method.

Trust-region methods are iterative processes. At each iteration k, they build a twice-continuously differentiable model m_k that we trust to adequately represent the objective function f inside some neighbourhood of the iterate x_k. This neighbourhood, called the *trust region*, is often represented by a ball in some norm (typically the Euclidean one), centred at the current iterate x_k and whose radius is Δ_k. The model is then (approximately) minimized inside the trust region $\mathcal{B}_k$, yielding a step s_k, which induces a *sufficient decrease* in the model. The trial point $x_k + s_k$ is then accepted as the next iterate x_{k+1} if the ratio

$$\rho_k \stackrel{\text{def}}{=} \frac{f(x_k) - f(x_k + s_k)}{m_k(x_k) - m_k(x_k + s_k)} \tag{2.61}$$

of achieved reduction (in the objective function f) to predicted reduction (in its local model m_k), is larger than a small positive constant η_1 (in which case the iteration k is said to be *successful*). The trust-region radius is then updated: it is decreased if the trial point is rejected (that is if $\rho_k < \eta_1$), and left unchanged or increased otherwise. This framework is formalized in Algorithm 2.11.

Algorithm 2.11 (Basic trust-region method: BTR)

Step 0. Initialization: An initial point x_0 and an initial trust-region radius $\Delta_0 > 0$ are given. The constants η_1, η_2, γ_1 and γ_2 are also given and satisfy

$$0 < \eta_1 \leqslant \eta_2 < 1 \text{ and } 0 < \gamma_1 \leqslant \gamma_2 < 1. \tag{2.62}$$

Compute $f(x_0)$ and set the iteration counter k to 0.

Step 1. Model definition: Select a twice-continuously differentiable model m_k defined in $\mathcal{B}_k$.

Step 2. Step calculation: Compute a step s_k that *sufficiently reduces* the model m_k and such that $x_k + s_k \in \mathcal{B}_k$.

Step 3. Acceptance of the trial point: Compute $f(x_k+s_k)$ and define

$$\rho_k = \frac{f(x_k) - f(x_k + s_k)}{m_k(x_k) - m_k(x_k + s_k)}. \tag{2.63}$$

If $\rho_k \geqslant \eta_1$, then set $x_{k+1} = x_k + s_k$; otherwise, set $x_{k+1} = x_k$.

Step 4. Trust-region update: Update its radius as follows:

$$\Delta_{k+1} \in \begin{cases} [\Delta_k, \infty) & \text{if } \rho_k \geqslant \eta_2, \\ [\gamma_2\Delta_k, \Delta_k) & \text{if } \rho_k \in [\eta_1, \eta_2), \\ [\gamma_1\Delta_k, \gamma_2\Delta_k) & \text{if } \rho_k < \eta_1. \end{cases} \tag{2.64}$$

Increment k by 1 and return to Step 1.

Implementation. — The model m_k is typically chosen as a (quasi-)Newton model, that is a quadratic model of the form (2.14), which is thus first-order coherent (see assumption A.2).

The problem consisting in minimizing the model m_k inside the trust region $\mathcal{B}_k$ is often called the *trust-region subproblem*. It may be solved in several ways (see Chapter 7 in Conn *et al.*, 2000, for a review), either exactly like with the method of Moré and Sorensen (1983), or approximately like with the truncated conjugate gradient method presented in Section 2.6.2. In any case, the method

used to solve the subproblem must enforce a sufficient decrease in the model, in the sense that

$$m_k(x_k) - m_k(x_k + s_k) \geqslant \kappa_{\text{mdc}} \|g_k\| \min\left[\frac{\|g_k\|}{1 + \|B_k\|}, \Delta_k\right] \tag{2.65}$$

for some constant $\kappa_{\text{mdc}} \in (0, 1)$. This inequality means that the decrease in the model achieved with the step s_k is at least a fraction of that achievable at the Cauchy point, that is the minimizer of the model m_k inside the trust region $\mathcal{B}_k$ along the direction $-g_k$ (see Dennis, 1978). Note that the Cauchy point is the first iterate of the truncated conjugate gradient method, which monotonically decreases the model value; hence the condition (2.65) is always fulfilled for this method.

Convergence. — As we thoroughly discuss the convergence of a variant of the basic trust-region method in Section 2.8.2, we only present briefly the convergence results for Algorithm 2.11, since both methods ultimately share the same properties. Under the assumptions that the model is first-order coherent, that the function and model Hessians norms are bounded above, and that a sufficient decrease in the model is achieved with each step, we then have the following results. First, if there are finitely many successful iterations, then the sequence of iterates becomes constant after a moment and converges to a first-order critical point (see Theorem 2.9, page 50). Second, the sequence of gradient norm at the iterates converges to 0 (see Theorem 2.11, page 50). Convergence to second-order critical points is also guaranteed under stronger assumptions on the models Hessians and with a slight modification in the updating rule (2.64) (see Theorems 2.13 to 2.15 on page 54).

We finally refer the reader to Conn *et al.* (2000) for a comprehensive coverage of this class of methods.

2.8 Retrospective trust-region method

We now present a variant of the trust-region method that we developed in collaboration with Fabian Bastin, Mélodie Mouffe, Philippe Toint and Dimitri Tomanos. This constitutes our first contribution.

2.8.1 Description

The retrospective trust-region method differs from the basic trust-region method presented in the previous section by the way the trust-region radius is updated. Two observations lead to this modification. First, the trust-region radius updating rule is completely based on the value of the ratio ρ_k in the basic version. However, the convergence theory of trust-region methods only requires that

the trust-region radius is shrunk after unsuccessful iterations (that is those for which $\rho_k < \eta_1$), leaving a complete freedom for the updates at successful iterations. Secondly, still in the basic version, the trust-region radius is updated from iteration k to iteration $k+1$ by considering how well the model m_k predicts the objective function value at iterate x_{k+1}. In retrospect, this might seem unnatural since the new radius Δ_{k+1} will determine the region in which a possibly updated model m_{k+1} is expected to predict the value of the objective function around x_{k+1}. We may therefore prefer to determine the radius Δ_{k+1} according to how well the new model m_{k+1} predicts the value of the objective function at x_k, thereby synchronizing the radius update with the change in models.

In the classical framework, the trust-region radius is updated at the end of each iteration: it is left unchanged or increased if the trial point is accepted (that is if $\rho_k \geqslant \eta_1$), and decreased otherwise. In this case, the new value Δ_{k+1} is chosen in the interval $[\gamma_0\|s_k\|, \gamma_1\|s_k\|]$ for some constants $0 < \gamma_0 < \gamma_1 \leqslant 1$. When ρ_k is negative, a quadratic fit of the model is used (as in Conn *et al.*, 2000, p. 783), to determine a tentative new radius whose purpose is to ensure that the next iteration is very successful in the sense that $\rho_{k+1} \geqslant \eta_2$ for some $\eta_2 \in (\eta_1, 1)$. This value is given by $\theta_k \Delta_k$, where

$$\theta_k \stackrel{\text{def}}{=} \frac{(1-\eta_2)\,\langle g_k, s_k\rangle}{(1-\eta_2)[f(x_k) + \langle g_k, s_k\rangle] + \eta_2 m_k(x_k+s_k) - f(x_k+s_k)}. \tag{2.66}$$

In the retrospective algorithm, the trust-region radius is thus updated after each successful iteration k (that is at the beginning of iteration $k+1$) on the basis of the *retrospective* ratio

$$\tilde{\rho}_{k+1} \stackrel{\text{def}}{=} \frac{f(x_{k+1}) - f(x_{k+1}-s_k)}{m_{k+1}(x_{k+1}) - m_{k+1}(x_{k+1}-s_k)} = \frac{f(x_k) - f(x_k+s_k)}{m_{k+1}(x_k) - m_{k+1}(x_k+s_k)}$$

of achieved to predicted changes, while continuing to use the ratio ρ_k to decide whether the trial iterate may be accepted. This method therefore distinguishes the two roles played by ρ_k in the classical algorithm: that of deciding acceptance of the trial iterate and that of determining the radius update. It also explicitly takes into account that m_{k+1}, not m_k, is used within the trust region of radius Δ_{k+1}. Thus, when the iterate has first been accepted, that is when $\rho_k \geqslant \eta_1$, we compute this radius by either increasing the current radius or leaving it unchanged if $\tilde{\rho}_k \geqslant \tilde{\eta}_1$ or decrease it otherwise. In this last case, it is again chosen in the interval $[\gamma_0\|s_k\|, \gamma_1\|s_k\|]$. Moreover, when $\tilde{\rho}_k$ is negative, a quadratic fit of the model is used as above to determine a tentative new radius which will make the next iteration very successful in the sense that $\tilde{\rho}_{k+1} \geqslant \tilde{\eta}_2$ for some $\tilde{\eta}_2 \in (\tilde{\eta}_1, 1)$. This value is given by $\tilde{\theta}_{k+1}\Delta_k$, where

$$\tilde{\theta}_{k+1} \stackrel{\text{def}}{=} \frac{-(1-\tilde{\eta}_2)\,\langle g_{k+1}, s_k\rangle}{(1-\tilde{\eta}_2)[f(x_{k+1}) - \langle g_{k+1}, s_k\rangle] + \tilde{\eta}_2 m_{k+1}(x_k) - f(x_k)}. \tag{2.67}$$

Notice that $\tilde{\theta}_{k+1}$ uses the gradient at the new point, rather than the old one as in (2.66).

This modification leads to the retrospective trust-region method described as Algorithm 2.12, in which we leave the precise definitions of the model (at Step 1) and of "sufficient reduction" (at Step 3) for the next section.

Algorithm 2.12 (Retrospective trust-region method: RTR)

Step 0. Initialization: An initial point x_0 and initial trust-region radius $\Delta_0 > 0$ are given. The constants η_1, $\tilde{\eta}_1$, $\tilde{\eta}_2$, γ_0, γ_1 and γ_2 are also given and satisfy

$$0 < \eta_1 < 1, \;\; 0 < \tilde{\eta}_1 \leqslant \tilde{\eta}_2 < 1 \;\text{ and }\; 0 < \gamma_0 < \gamma_1 \leqslant 1 \leqslant \gamma_2. \tag{2.68}$$

Compute $f(x_0)$ and set the iteration counter k to 0.

Step 1. Model definition: Select a twice-continuously differentiable model m_k defined in $\mathcal{B}_k$.

Step 2. Retrospective trust-region radius update: If $k = 0$, go to Step 3. If $x_k = x_{k-1}$, then choose

$$\Delta_k = \begin{cases} \gamma_1 \|s_{k-1}\| & \text{if } \rho_{k-1} \in [0, \eta_1), \\ \min[\gamma_1 \|s_{k-1}\|, \max[\gamma_0, \theta_{k-1}]\Delta_{k-1}] & \text{if } \rho_{k-1} < 0, \end{cases} \tag{2.69}$$

where θ_{k-1} is defined in (2.66). Else, define

$$\tilde{\rho}_k = \frac{f(x_{k-1}) - f(x_k)}{m_k(x_{k-1}) - m_k(x_k)} \tag{2.70}$$

and choose

$$\Delta_k = \begin{cases} \max[\gamma_2 \|s_{k-1}\|, \Delta_{k-1}] & \text{if } \tilde{\rho}_k \geqslant \tilde{\eta}_2, \\ \Delta_{k-1} & \text{if } \tilde{\rho}_k \in [\tilde{\eta}_1, \tilde{\eta}_2), \\ \gamma_1 \|s_{k-1}\| & \text{if } \tilde{\rho}_k \in [0, \tilde{\eta}_1), \\ \min[\gamma_1 \|s_{k-1}\|, \max[\gamma_0, \tilde{\theta}_k]\Delta_{k-1}] & \text{if } \tilde{\rho}_k < 0, \end{cases} \tag{2.71}$$

where $\tilde{\theta}_k$ is defined in (2.67).

Step 3. Step calculation: Compute a step s_k that *sufficiently reduces* the model m_k and such that $x_k + s_k \in \mathcal{B}_k$.

Step 4. Acceptance of the trial point: Compute $f(x_k + s_k)$ and define

$$\rho_k = \frac{f(x_k) - f(x_k + s_k)}{m_k(x_k) - m_k(x_k + s_k)}. \tag{2.72}$$

If $\rho_k \geqslant \eta_1$, then define $x_{k+1} = x_k + s_k$ and compute $\nabla f(x_{k+1})$; otherwise define $x_{k+1} = x_k$. Increment k by 1 and return to Step 1.

2.8.2 Convergence properties

We now investigate the convergence properties of our algorithm. Since it can be considered as a variant of the basic trust-region method of Conn *et al.* (2000), we expect similar results and significant similarities in their proofs. In what follows, we have attempted to be explicit on the assumptions and properties, but to refer to Chapter 6 of this reference whenever possible. Our assumptions are identical to those used for the basic trust-region method.

A.1 The Hessian of the objective function $\nabla^2 f$ is uniformly bounded, *i.e.* there exists a positive constant κ_{ufh} such that, for all $x \in \mathbb{R}^n$,

$$\|\nabla^2 f(x)\| \leqslant \kappa_{\text{ufh}}. \tag{2.73}$$

A.2 The model m_k is first-order coherent with the function f at each iteration x_k, *i.e.* their values and gradients are equal at x_k for all k:

$$m_k(x_k) = f(x_k) \quad \text{and} \quad g_k \stackrel{\text{def}}{=} \nabla m_k(x_k) = \nabla f(x_k).$$

A.3 The of the model $\nabla^2 m_k$ is uniformly bounded, *i.e.* there exists a constant $\kappa_{\text{umh}} \geqslant 1$ such that, for all $x \in \mathbb{R}^n$ and for all k,

$$\|\nabla^2 m_k(x)\| \leqslant \kappa_{\text{umh}} - 1.$$

A.4 The decrease on the model m_k is at least as much as a fraction of that obtained at the Cauchy point, *i.e.* there exists a constant $\kappa_{\text{mdc}} \in (0, 1)$ such that, for all k,

$$m_k(x_k) - m_k(x_k + s_k) \geqslant \kappa_{\text{mdc}} \|g_k\| \min\left[\frac{\|g_k\|}{\beta_k}, \Delta_k\right]$$

with $\beta_k \stackrel{\text{def}}{=} 1 + \max_{x \in \mathcal{B}_k} \|\nabla^2 m_k(x)\|$.

Note that the assumption A.4 specifies the notion of *sufficient reduction* used in Step 3 of our algorithm, while the choice of m_k in Step 1 is limited by A.2 and A.3. We also note that $s_k \neq 0$ whenever $g_k \neq 0$ because of A.4.

2.8.2.1 Convergence to first-order critical points

In this section, we prove that the retrospective trust-region algorithm is globally convergent to first-order critical points, in the sense that every limit point x_* of the sequence of iterates (x_k) produced by Algorithm 2.12 satisfies

$$\nabla f(x_*) = 0$$

irrespective of the choice of the starting point x_0 and initial trust-region radius Δ_0.

We first give a bound on the error between the true objective function f and its current model m_k at the previous iterate x_{k-1}.

Theorem 2.5 *Suppose that A.1–A.3 hold. Then we have that*

$$|f(x_k) - m_{k-1}(x_k)| \leqslant \kappa_{\text{ubh}} \Delta_{k-1}^2 \tag{2.74}$$

and, if iteration $k-1$ is successful, that

$$|f(x_{k-1}) - m_k(x_{k-1})| \leqslant \kappa_{\text{ubh}} \Delta_{k-1}^2 \tag{2.75}$$

where

$$\kappa_{\text{ubh}} \stackrel{\text{def}}{=} \max[\kappa_{\text{ufh}}, \kappa_{\text{umh}}]. \tag{2.76}$$

Proof. — The bound (2.74) directly results from Theorem 6.4.1 in Conn *et al.* (2000). We thus only prove (2.75). Because the objective function and the model are $\mathcal{C}^2$ functions, we may apply the mean-value theorem 1.1 on the objective function f and on the model m_k, and obtain from $x_{k-1} = x_k - s_{k-1}$ that

$$f(x_{k-1}) = f(x_k) - \langle s_{k-1}, \nabla f(x_k) \rangle + \tfrac{1}{2} \left\langle s_{k-1}, \nabla^2 f(\xi_k) s_{k-1} \right\rangle \tag{2.77}$$

$$m_k(x_{k-1}) = m_k(x_k) - \langle s_{k-1}, g_k \rangle + \tfrac{1}{2} \left\langle s_{k-1}, \nabla^2 m_k(\zeta_k) s_{k-1} \right\rangle \tag{2.78}$$

for some ξ_k, ζ_k in the segment $[x_{k-1}, x_k]$.

Because of A.2, the objective function f and the model m_k have the same value and gradient at x_k. Thus, subtracting (2.78) from (2.77) and taking absolute values yields that

$$\begin{aligned} |f(x_{k-1}) - m_k(x_{k-1})| &= \tfrac{1}{2} \left| \left\langle s_{k-1}, \nabla^2 f(\xi_k) s_{k-1} \right\rangle - \left\langle s_{k-1}, \nabla^2 m_k(\zeta_k) s_{k-1} \right\rangle \right| \\ &\leqslant \tfrac{1}{2} \left[\left| \left\langle s_{k-1}, \nabla^2 f(\xi_k) s_{k-1} \right\rangle \right| + \left| \left\langle s_{k-1}, \nabla^2 m_k(\zeta_k) s_{k-1} \right\rangle \right| \right] \\ &\leqslant \tfrac{1}{2} (\kappa_{\text{ufh}} + \kappa_{\text{umh}} - 1) \|s_{k-1}\|^2 \\ &\leqslant \tfrac{1}{2} (\kappa_{\text{ufh}} + \kappa_{\text{umh}} - 1) \Delta_{k-1}^2, \end{aligned} \tag{2.79}$$

where we successively used the triangle inequality, the Cauchy-Schwarz inequality, the induced matrix norm properties, A.1, A.3, and the fact that $x_k \in \mathcal{B}_{k-1}$ implies that $\|s_{k-1}\| \leqslant \Delta_{k-1}$. So (2.75) clearly holds. ∎

Thus the analog of Theorem 6.4.1 of Conn *et al.* (2000) holds in our case, where we replace the forward difference $f(x_{k+1}) - m_k(x_{k+1})$ by its retrospective variant $f(x_{k-1}) - m_k(x_{k-1})$.

As our new ratio $\tilde{\rho}_k$ uses the reduction in m_k instead of the reduction in m_{k-1}, we are interested in a bound on their difference, which is provided by this next result.

Lemma 2.6 *Suppose that A.1–A.3 hold. Then we have that, for every successful iteration $k-1$,*

$$| \, [m_{k-1}(x_{k-1}) - m_{k-1}(x_k)] - [m_k(x_{k-1}) - m_k(x_k)] \, | \leqslant 2\kappa_{\text{ubh}} \Delta_{k-1}^2. \tag{2.80}$$

Proof. — Using the model differentiability, we apply the mean-value theorem on the model m_{k-1}, and we obtain that

$$m_{k-1}(x_k) = m_{k-1}(x_{k-1}) + \langle s_{k-1}, g_{k-1} \rangle + \tfrac{1}{2} \langle s_{k-1}, \Psi_{k-1} s_{k-1} \rangle \tag{2.81}$$

where $\Psi_{k-1} = \nabla^2 m_{k-1}(\psi_{k-1})$ for some point ψ_{k-1} in the segment $[x_{k-1}, x_k]$. Remember that equation (2.78) in the previous proof gives that

$$m_k(x_{k-1}) = m_k(x_k) - \langle s_{k-1}, g_k \rangle + \tfrac{1}{2} \langle s_{k-1}, Z_k s_{k-1} \rangle \tag{2.82}$$

where $Z_k = \nabla^2 m_k(\zeta_k)$ for some point ζ_k in the segment $[x_{k-1}, x_k]$. Substituting (2.81) and (2.82) inside the left-hand side of (2.80), and using A.3, the triangle inequality, the Cauchy-Schwarz inequality, and the induced matrix norm properties yield that

$$\begin{aligned} &\big|[m_{k-1}(x_{k-1}) - m_{k-1}(x_k)] - [m_k(x_{k-1}) - m_k(x_k)]\big| \\ &\quad = \Big| - \langle s_{k-1}, g_{k-1} - g_k \rangle - \tfrac{1}{2} \left(\langle s_{k-1}, \Psi_{k-1} s_{k-1} \rangle + \langle s_{k-1}, Z_k s_{k-1} \rangle \right) \Big| \\ &\quad \leqslant \|s_{k-1}\| \cdot \|g_{k-1} - g_k\| + \kappa_{\text{umh}} \|s_{k-1}\|^2. \end{aligned} \tag{2.83}$$

Now observe that, because of A.2, $\|g_{k-1} - g_k\| = \|\nabla f(x_{k-1}) - \nabla f(x_k)\|$. We then apply the mean-value theorem on ∇f and obtain that

$$\nabla f(x_k) = \nabla f(x_{k-1}) + \int_0^1 \nabla^2 f(x_{k-1} + \alpha s_{k-1}) s_{k-1} \, d\alpha. \tag{2.84}$$

Thus the Cauchy-Schwarz inequality and A.1 give that

$$\|g_{k-1} - g_k\| \leqslant \int_0^1 \|\nabla^2 f(x_{k-1} + \alpha s_{k-1})\| \cdot \|s_{k-1}\| \, d\alpha \tag{2.85}$$

$$\leqslant \int_0^1 \kappa_{\text{ufh}} \|s_{k-1}\| \, d\alpha \leqslant \kappa_{\text{ufh}} \|s_{k-1}\|. \tag{2.86}$$

Substituting this bound in (2.83), we obtain that

$$\begin{aligned} |\,[m_{k-1}(x_{k-1}) - m_{k-1}(x_k)] - [m_k(x_{k-1}) - m_k(x_k)]\,| &\leqslant (\kappa_{\rm ufh} + \kappa_{\rm umh})\|s_{k-1}\|^2 \\ &\leqslant 2\kappa_{\rm ubh}\Delta_{k-1}^2 \end{aligned}$$

where we finally use (2.76), and the fact that $x_k \in \mathcal{B}_{k-1}$. ∎

We conclude from this result that the denominators in the expression of $\tilde{\rho}_k$ and ρ_{k-1} differ by a quantity which is of the same order as the error between the model and the objective function. Using this observation, we are now capable of showing that the iteration must be successful if the radius is sufficiently small compared to the gradient, and also that the trust-region radius has to increase in this case.

Theorem 2.7 *Suppose that A.1–A.4 hold. Suppose furthermore that* $g_{k-1} \neq 0$ *and that*

$$\Delta_{k-1} \leqslant \min\left[1 - \eta_1, \frac{(1-\tilde{\eta}_2)}{(3-2\tilde{\eta}_2)}\right] \frac{\kappa_{\rm mdc}}{\kappa_{\rm ubh}} \|g_{k-1}\|. \tag{2.87}$$

Then iteration $k-1$ *is successful and*

$$\Delta_k \geqslant \Delta_{k-1}. \tag{2.88}$$

Proof. — We first apply Theorem 6.4.2 of Conn *et al.* (2000) to deduce that iteration $k-1$ is successful and thus that $x_k = x_{k-1} + s_{k-1} \neq x_{k-1}$. Observe now that the constants $\tilde{\eta}_2$ and $\kappa_{\rm mdc}$ lie in the interval $(0, 1)$, which implies that

$$\frac{(1-\tilde{\eta}_2)}{(3-2\tilde{\eta}_2)} < \frac{1}{2} < 1 \quad \text{and thus} \quad \kappa_{\rm mdc}\frac{(1-\tilde{\eta}_2)}{(3-2\tilde{\eta}_2)} < 1. \tag{2.89}$$

The conditions (2.87), (2.89), and (2.76), combined with the definition of β_{k-1} in A.4 imply that

$$\Delta_{k-1} < \frac{\|g_{k-1}\|}{\kappa_{\rm ubh}} < \frac{\|g_{k-1}\|}{\beta_{k-1}}. \tag{2.90}$$

As a consequence, A.4 immediately gives that

$$\begin{aligned} m_{k-1}(x_{k-1}) - m_{k-1}(x_k) &\geqslant \kappa_{\rm mdc}\|g_{k-1}\| \min\left[\frac{\|g_{k-1}\|}{\beta_{k-1}}, \Delta_{k-1}\right] \\ &= \kappa_{\rm mdc}\|g_{k-1}\|\Delta_{k-1}. \end{aligned} \tag{2.91}$$

On the other hand, we may apply Lemma 2.6 and use the triangle inequality to obtain that

$$\begin{aligned}\big|m_{k-1}(x_{k-1}) - m_{k-1}(x_k)\big| - \big|m_k(x_{k-1}) - m_k(x_k)\big| \\ \leqslant \big|\big[m_{k-1}(x_{k-1}) - m_{k-1}(x_k)\big] - \big[m_k(x_{k-1}) - m_k(x_k)\big]\big| \\ \leqslant 2\kappa_{\rm ubh}\Delta_{k-1}^2\end{aligned}$$

and therefore, with (2.91), that

$$\begin{aligned}|m_k(x_{k-1}) - m_k(x_k)| &\geqslant |m_{k-1}(x_{k-1}) - m_{k-1}(x_k)| - 2\kappa_{\rm ubh}\Delta_{k-1}^2 \\ &\geqslant \kappa_{\rm mdc}\|g_{k-1}\|\Delta_{k-1} - 2\kappa_{\rm ubh}\Delta_{k-1}^2. \end{aligned} \tag{2.92}$$

Note moreover that equation (2.87) implies that

$$(3 - 2\tilde{\eta}_2)\kappa_{\rm ubh}\Delta_{k-1} \leqslant (1 - \tilde{\eta}_2)\kappa_{\rm mdc}\|g_{k-1}\|,$$

and thus that

$$(1 - \tilde{\eta}_2)(\kappa_{\rm mdc}\|g_{k-1}\| - 2\kappa_{\rm ubh}\Delta_{k-1}) \geqslant \kappa_{\rm ubh}\Delta_{k-1} > 0. \tag{2.93}$$

We finally may apply Theorem 2.5 and deduce from A.2, (2.75), (2.92) and (2.93) that

$$|\tilde{\rho}_k - 1| = \left|\frac{f(x_{k-1}) - m_k(x_{k-1})}{m_k(x_{k-1}) - m_k(x_k)}\right| \leqslant \frac{\kappa_{\rm ubh}\Delta_{k-1}}{\kappa_{\rm mdc}\|g_{k-1}\| - 2\kappa_{\rm ubh}\Delta_{k-1}} \leqslant 1 - \tilde{\eta}_2. \tag{2.94}$$

Therefore, $\tilde{\rho}_k \geqslant \tilde{\eta}_2$ and (2.71) then ensures that (2.88) holds. ∎

It is therefore guaranteed that the trust-region radius can not be decreased indefinitely if the current iterate is not near criticality. This is ensured by the next theorem.

Theorem 2.8 *Suppose that A.1–A.4 hold. Suppose furthermore that there exists a constant $\kappa_{\rm lbg}$ such that $\|g_k\| \geqslant \kappa_{\rm lbg}$ for all k. Then there is a constant $\kappa_{\rm lbd}$ such that*

$$\Delta_k \geqslant \kappa_{\rm lbd} \tag{2.95}$$

for all k.

Proof. — The proof is the same as for Theorem 6.4.3 in Conn *et al.* (2000) except that

$$\kappa_{\rm lbd} = \min\left[1 - \eta_1, \frac{(1 - \tilde{\eta}_2)}{(3 - 2\tilde{\eta}_2)}\right]\frac{\gamma_1\kappa_{\rm mdc}\kappa_{\rm lbg}}{\kappa_{\rm ubh}}.$$

∎

From here on, the proof for the basic trust-region method applies without change. We first deduce the global convergence of the algorithm to first-order critical points when it generates only finitely many successful iterations.

Theorem 2.9 *Suppose that A.1–A.4 hold. Suppose furthermore that there are only finitely many successful iterations. Then $x_k = x_*$ for all sufficiently large k and x_* is first-order critical.*

Proof. — The same argument as in Theorem 6.4.4 in Conn *et al.* (2000) may be applied since the radius update is identical to that of the basic trust region method for unsuccessful iterations. ∎

Finally, the next two results ensure the global convergence of the algorithm to first-order critical points, by showing in a first step that at least one accumulation point of the iterates sequence is first-order critical.

Theorem 2.10 *Suppose that A.1–A.4 hold. Then one has that*

$$\liminf_{k\to\infty} \|\nabla f(x_k)\| = 0. \tag{2.96}$$

Proof. — See Theorem 6.4.5 in Conn *et al.* (2000). ∎

As for the basic trust-region method, this can be extended to show that all limit points are first-order critical.

Theorem 2.11 *Suppose that A.1–A.4 hold. Then one has that*

$$\lim_{k\to\infty} \|\nabla f(x_k)\| = 0. \tag{2.97}$$

Proof. — See Theorem 6.4.6 in Conn *et al.* (2000). ∎

2.8.2.2 Convergence to second-order critical points

We now investigate the possibility to exploit second-order information on the objective function, with the aim of ensuring convergence to second-order critical points. Of course, we need to clarify what we precisely mean by "second-order information". We therefore introduce the following additional assumptions:

A.5 The model is asymptotically second-order coherent with the objective function near first-order critical points, *i.e.*

$$\lim_{k\to\infty} \|\nabla^2 f(x_k) - \nabla^2 m_k(x_k)\| = 0 \text{ whenever } \lim_{k\to\infty} \|g_k\| = 0.$$

A.6 The Hessian of every model m_k is Lipschitz continuous, that is, there exists a constant $\kappa_{\rm lch}$ such that, for all k,

$$\|\nabla^2 m_k(x) - \nabla^2 m_k(y)\| \leqslant \kappa_{\rm lch}\|x - y\|$$

for all $x, y \in \mathcal{B}_k$.

A.7 If the smallest eigenvalue τ_k of the Hessian of the model m_k at x_k is negative, then

$$m_k(x_k) - m_k(x_k + s_k) \geqslant \kappa_{\rm sod}|\tau_k| \min(\tau_k^2, \Delta_k^2)$$

for some constant $\kappa_{\rm sod} \in (0, \frac{1}{2})$.

These assumptions are identical to those used in Sections 6.5 and 6.6 of Conn *et al.* (2000) for the basic trust-region method. In fact, the second-order convergence properties of the retrospective trust-region method also turn out to be exactly the same as those of the basic trust-region method, and their proofs can essentially been borrowed from this case, with the exception of Lemma 6.5.3. We therefore need to present a proof of that particular result for the new method. As we indicate below, all other results generalize without change and we only mention them for the sake of clarity.

In our analog of Lemma 6.5.3, we assume that the model reduction is eventually significant in the sense that it is at least of the same order as the error between the model and the objective function. We then show that the trust-region radius becomes asymptotically irrelevant if the steps tend to zero.

Lemma 2.12 *Suppose that A.1–A.3, and A.5 hold. Suppose also that there exists a sequence* (k_i) *and a constant* $\kappa_{\rm mqd} > 0$ *such that*

$$m_{k_i}(x_{k_i}) - m_{k_i}(x_{k_i} + s_{k_i}) \geqslant \kappa_{\rm mqd}\|s_{k_i}\|^2 > 0 \tag{2.98}$$

for all i *sufficiently large. Finally, suppose that*

$$\lim_{i\to\infty} \|s_{k_i}\| = 0.$$

Then iteration k_i *is successful and*

$$\tilde{\rho}_{k_i+1} \geqslant \tilde{\eta}_2 \quad \text{and} \quad \Delta_{k_i+1} \geqslant \Delta_{k_i} \tag{2.99}$$

for i *sufficiently large.*

Proof. — We first apply Lemma 6.5.3 of Conn *et al.* (2000) to deduce that every iteration k_i is successful for i sufficiently large. Now, consider k_i one such iteration, and $k_i^+ = k_i + 1$ the next iteration. The equations (2.77) and (2.78) imply that for some $\xi_{k_i^+}$ and $\zeta_{k_i^+}$ in the segment $[x_{k_i}, x_{k_i^+}]$,

$$\begin{aligned}
|\tilde{\rho}_{k_i^+} - 1| &= \left| \frac{f(x_{k_i}) - m_{k_i^+}(x_{k_i})}{m_{k_i^+}(x_{k_i}) - m_{k_i^+}(x_{k_i^+})} \right| \\
&= \left| \frac{\langle s_{k_i}, \nabla^2 f(\xi_{k_i^+}) s_{k_i} \rangle - \langle s_{k_i}, \nabla^2 m_{k_i^+}(\zeta_{k_i^+}) s_{k_i} \rangle}{-\langle s_{k_i}, g_{k_i^+} \rangle + \frac{1}{2} \langle s_{k_i}, \nabla^2 m_{k_i^+}(\zeta_{k_i^+}) s_{k_i} \rangle} \right| \\
&\leqslant \frac{\|s_{k_i}\|^2 \cdot \|\nabla^2 f(\xi_{k_i^+}) - \nabla^2 m_{k_i^+}(\zeta_{k_i^+})\|}{\left| -\langle s_{k_i}, g_{k_i^+} \rangle + \frac{1}{2} \langle s_{k_i}, \nabla^2 m_{k_i^+}(\zeta_{k_i^+}) s_{k_i} \rangle \right|}
\end{aligned} \tag{2.100}$$

where we also used the Cauchy-Schwarz inequality. By substituting $g_{k_i^+} = \nabla f(x_{k_i^+})$ (because of A.2) with its expression in (2.84), the denominator D of the latter fraction can be rewritten as

$$D = \left| -\left\langle s_{k_i}, g_{k_i} + \int_0^1 \nabla^2 f(x_{k_i} + \alpha s_{k_i}) s_{k_i} \, d\alpha \right\rangle + \tfrac{1}{2} \left\langle s_{k_i}, \nabla^2 m_{k_i^+}(\zeta_{k_i^+}) s_{k_i} \right\rangle \right|.$$

Then, replacing $-\langle s_{k_i}, g_{k_i} \rangle$ by its expression in (2.81), we obtain

$$\begin{aligned}
D = \Bigg| & m_{k_i}(x_{k_i}) - m_{k_i}(x_{k_i^+}) + \tfrac{1}{2} \left\langle s_{k_i}, \nabla^2 m_{k_i}(\psi_{k_i}) s_{k_i} \right\rangle \\
& + \tfrac{1}{2} \left\langle s_{k_i}, \nabla^2 m_{k_i^+}(\zeta_{k_i^+}) s_{k_i} \right\rangle - \left\langle s_{k_i}, \int_0^1 \nabla^2 f(x_{k_i} + \alpha s_{k_i}) s_{k_i} \, d\alpha \right\rangle \Bigg|
\end{aligned}$$

for some ψ_{k_i} in the segment $[x_{k_i}, x_{k_i^+}]$. The triangle inequality, properties of the integral, (2.98), and Cauchy-Schwarz inequality give therefore the following lower bound on D:

$$\begin{aligned}
D \quad \geqslant \quad & \left| m_{k_i}(x_{k_i}) - m_{k_i}(x_{k_i^+}) \right| \\
& - \tfrac{1}{2} \Bigg| \left\langle s_{k_i}, \int_0^1 [\nabla^2 f(x_{k_i} + \alpha s_{k_i}) - \nabla^2 m_{k_i}(\psi_{k_i})] s_{k_i} \, d\alpha \right\rangle \\
& \qquad + \left\langle s_{k_i}, \int_0^1 [\nabla^2 f(x_{k_i} + \alpha s_{k_i}) - \nabla^2 m_{k_i^+}(\zeta_{k_i^+})] s_{k_i} \, d\alpha \right\rangle \Bigg| \\
\geqslant \quad & \kappa_{\text{mqd}} \|s_{k_i}\|^2 - \tfrac{1}{2} \|s_{k_i}\| \int_0^1 \|\nabla^2 f(x_{k_i} + \alpha s_{k_i}) - \nabla^2 m_{k_i}(\psi_{k_i})\| \cdot \|s_{k_i}\| \, d\alpha \\
& \qquad - \tfrac{1}{2} \|s_{k_i}\| \int_0^1 \|\nabla^2 f(x_{k_i} + \alpha s_{k_i}) - \nabla^2 m_{k_i^+}(\zeta_{k_i^+})\| \cdot \|s_{k_i}\| \, d\alpha \\
\geqslant \quad & \|s_{k_i}\|^2 (\kappa_{\text{mqd}} - \tfrac{1}{2} \epsilon_i)
\end{aligned} \tag{2.101}$$

where

$$\epsilon_i \stackrel{\text{def}}{=} \int_0^1 \|\nabla^2 f(x_{k_i} + \alpha s_{k_i}) - \nabla^2 m_{k_i}(\psi_{k_i})\| \, d\alpha + \int_0^1 \|\nabla^2 f(x_{k_i} + \alpha s_{k_i}) - \nabla^2 m_{k_i^+}(\zeta_{k_i^+})\| \, d\alpha. \quad (2.102)$$

The triangle inequality now implies that

$$\begin{aligned} \|\nabla^2 f(x_{k_i} + \alpha s_{k_i}) - \nabla^2 m_{k_i}(\psi_{k_i})\| \leqslant & \|\nabla^2 f(x_{k_i} + \alpha s_{k_i}) - \nabla^2 f(x_{k_i})\| \\ & + \|\nabla^2 f(x_{k_i}) - \nabla^2 m_{k_i}(x_{k_i})\| \\ & + \|\nabla^2 m_{k_i}(x_{k_i}) - \nabla^2 m_{k_i}(\psi_{k_i})\| \end{aligned} \quad (2.103)$$

and, similarly, that

$$\begin{aligned} \|\nabla^2 f(x_{k_i} + \alpha s_{k_i}) - \nabla^2 m_{k_i^+}(\zeta_{k_i^+})\| \leqslant & \|\nabla^2 f(x_{k_i} + \alpha s_{k_i}) - \nabla^2 f(x_{k_i^+})\| \\ & + \|\nabla^2 f(x_{k_i^+}) - \nabla^2 m_{k_i^+}(x_{k_i^+})\| \\ & + \|\nabla^2 m_{k_i^+}(x_{k_i^+}) - \nabla^2 m_{k_i^+}(\zeta_{k_i^+})\|. \end{aligned} \quad (2.104)$$

Since we now observe that

$$\begin{aligned} \|(x_{k_i} + \alpha s_{k_i}) - x_{k_i}\| \leqslant \|s_{k_i}\|, \qquad & \|\psi_{k_i} - x_{k_i}\| \leqslant \|s_{k_i}\|, \\ \|(x_{k_i} + \alpha s_{k_i}) - x_{k_i^+}\| \leqslant \|s_{k_i}\|, \qquad & \|\zeta_{k_i^+} - x_{k_i^+}\| \leqslant \|s_{k_i}\|, \end{aligned}$$

we may deduce that both

$$\|\nabla^2 f(x_{k_i} + \alpha s_{k_i}) - \nabla^2 m_{k_i}(\psi_{k_i})\| \text{ and } \|\nabla^2 f(x_{k_i} + \alpha s_{k_i}) - \nabla^2 m_{k_i^+}(\zeta_{k_i^+})\|$$

converge to zero with $\|s_{k_i}\|$ because the first and third terms of the right-hand side of (2.103) and (2.104) tend to zero by continuity of the the objective function's and model's Hessians, and because the middle term in the right-hand side of these inequalities also converges to zero because of A.5 and Theorem 2.11. As a consequence, $\epsilon_i \leqslant \kappa_{\text{mqd}}$ when i is sufficiently large, and therefore, combining (2.100) and (2.101), and using the triangle inequality, we obtain

$$\begin{aligned} |\tilde{\rho}_{k_i^+} - 1| &\leqslant \frac{2}{\kappa_{\text{mqd}}} \|\nabla^2 f(\xi_{k_i^+}) - \nabla^2 m_{k_i^+}(\zeta_{k_i^+})\| \\ &\leqslant \frac{2}{\kappa_{\text{mqd}}} \Big[\|\nabla^2 f(\xi_{k_i^+}) - \nabla^2 f(x_{k_i^+})\| \\ &\qquad + \|\nabla^2 f(x_{k_i^+}) - \nabla^2 m_{k_i^+}(x_{k_i^+})\| \\ &\qquad + \|\nabla^2 m_{k_i^+}(x_{k_i^+}) - \nabla^2 m_{k_i^+}(\zeta_{k_i^+})\| \Big] \end{aligned} \quad (2.105)$$

By the same reasoning as for (2.103)–(2.104), the right-hand side of (2.105) tends to zero when i goes to infinity, and $\tilde{\rho}_{k_i^+}$ therefore tends to 1. It is thus larger than $\tilde{\eta}_2 < 1$ for i sufficiently large and (2.99) follows. ∎

As in Lemma 6.5.4 of Conn *et al.* (2000), we may apply this result to the entire sequence of iterates and deduce that all iterations are eventually successful and the trust-region radius bounded away from zero.

From here on, the theory in Conn *et al.* (2000) generalizes without significant change, yielding the following results.

Theorem 2.13 *Suppose that A.1–A.5 hold and that x_{k_i} is a subsequence of the iterates generated by Algorithm RTR converging to a first-order critical point x_* where the Hessian of the objective function $\nabla^2 f(x_*)$ is positive definite. Suppose furthermore that $s_k \neq 0$ for all k sufficiently large. Then the complete sequence of iterates converges to x_*, all iterations are eventually very successful, and the trust-region radius Δ_k is bounded away from zero.*

Proof. — See Theorem 6.5.5 in Conn *et al.* (2000). ∎

We now prove that if the sequence of iterates remains in a compact set, then the existence of at least one second-order critical accumulation point is guaranteed.

Theorem 2.14 *Suppose that A.1–A.7 hold and that all iterates remain in some compact set. Then there exists at least one limit point x_* of the sequence of iterates x_k produced by Algorithm RTR, which is second-order critical.*

Proof. — See Theorem 6.6.5 in Conn *et al.* (2000). ∎

By just strengthening the radius update rule by requiring that

$$\text{if } \tilde{\rho}_k \geqslant \tilde{\eta}_2 \text{ and } \Delta_k \leqslant \Delta_{\max}, \text{ then } \Delta_{k+1} \in [\gamma_3 \Delta_k, \gamma_4 \Delta_k] \tag{2.106}$$

for some $\gamma_4 \geqslant \gamma_3 > 1$ and some $\Delta_{\max} > 0$, we moreover obtain the second-order criticality of any limit point of the sequence of iterates generated by Algorithm RTR.

Theorem 2.15 *Suppose that A.1–A.7, and* (2.106) *hold and let x_* be any limit point of the sequence of iterates. Then x_* is a second-order critical point.*

Proof. — See Theorem 6.6.8 in Conn *et al.* (2000). ∎

Thus the retrospective trust-region algorithm shares all the (interesting) convergence properties of the basic trust-region method under the same assumptions. We conclude this theory section by noting that the above convergence results are still valid if one replaces the Euclidean norm by any (possibly iteration dependent) uniformly equivalent norm, thereby allowing problem scaling and preconditioning.

2.8.3 Numerical experiments

We now consider the numerical behaviour of the new algorithm, in comparison with the basic trust-region algorithm BTR (see page 116 of Conn *et al.*, 2000). We test both algorithms on all of the 146 unconstrained problems of the CUTEr collection (see Gould, Orban and Toint, 2003*a*). For the problems whose dimension may be changed, we chose a reasonably small value (from 2 to 500) in order not to overload the CUTEr interface with the MATLAB® code. The starting points are the standard ones provided by the CUTEr library.

For the basic algorithm, the trust-region radius update was implemented by using the rule proposed in Conn *et al.* (2000), p. 783:

$$\Delta_{k+1} = \begin{cases} \max[\gamma_2\|s_k\|, \Delta_k] & \text{if } \rho_k \geqslant \eta_2, \\ \Delta_k & \text{if } \rho_k \in [\eta_1, \eta_2), \\ \gamma_1\|s_k\| & \text{if } \rho_k \in [0, \eta_1), \\ \min[\gamma_1\|s_k\|, \max[\gamma_0, \theta_k]\Delta_k] & \text{if } \rho_k < 0, \end{cases}$$

where γ_0 is fixed at 0.0625, γ_1 at 0.25, γ_2 at 2.5, η_1 at 0.05 and η_2 at 0.9 and where θ_k is given by (2.66). To avoid biasing the comparison, we have decided to make as few adaptations as possible to that rule in our retrospective variant (*i.e.* Step 2 in Algorithm 2.12). Thus, if iteration k is unsuccessful, *i.e.* $\rho_k < \eta_1$ and consequently $x_k = x_{k+1}$, we also decrease the trust-region radius using the above rule. If, on the contrary, iteration k is successful, *i.e.* $\rho_k \geqslant \eta_1$, the trust-region radius is updated using the procedure described in Step 2 of Algorithm 2.12 where we choose the same values as above for γ_0, γ_1 and γ_2, and take $\tilde{\eta}_1 = \eta_1 = 0.05$ and $\tilde{\eta}_2 = \eta_2 = 0.9$. The model was chosen, in both cases, to be the exact Taylor's series truncated to second-order, and the minimizer of the model inside the trust region, was computed either exactly using the Moré-Sorensen algorithm (see Moré and Sorensen, 1983) or approximately using the Steihaug-Toint algorithm (see Subsection 2.6.2). In this case, the conjugate gradient iterations are stopped if the trust-region boundary is met or as soon as the models' gradient satisfies the condition

$$\|\nabla_x m_k(x_k + s)\| \leqslant \|g_k\| \min[0.1, \|g_k\|^{1/2}].$$

This condition is advised by Conn *et al.* (2000, Section 7.5.1), and ensures that the norm of the model gradient has been reduced to a small fraction of its initial value. The second term of the minimum increases the required accuracy on the subproblem solution as the trust-region algorithm approaches a first-order critical point. We considered that the trust-region method converged when the Euclidean norm of the gradient became smaller than 10^{-5}. Failure was declared if the algorithm did not converge within the maximum number of 50,000 iterations.

We chose to compare the number of iterations to achieve convergence instead of the CPU time or number of function evaluations. Indeed, the cost per iteration is the same for both algorithms and they both evaluate the objective function once per iteration and compute one gradient at every successful iteration. Moreover, timings in MATLAB® are often difficult to interpret.

All runs were performed in MATLAB® v. 7.1.0.183 (R14) Service Pack 3 on a 3.2 Ghz Intel® single-core processor computer with 2 GB of RAM. Figure 2.2 represents the comparison by a performance profile (see Section 1.5) of the number of iterations of the two algorithms. In this figure, we have only kept the problems for which both algorithms converged to the same local solution (we excluded `BIGGS6`, `BROYDN7D`, `CHAINWOO`, `FLETCHBV`, `LOGHAIRY`, `MEYER3`, `NONCVXU2`, `NONCVXUN`, `SENSORS`, `TOINTGSS` and `VIBRBEAM`). If the subproblem is solved approximately, both algorithms failed on `PALMER1C`, `SBRYBND`, `SCOSINE`, `SCURLY10`, `SCURLY20` and `SCURLY30`. Moreover, RTR failed on `FLETCBV3`, which was solved by BTR. On the other hand, if the subproblem is solved exactly, both algorithms failed on `FLETCBV3` and BTR failed on `SCOSINE`, which was solved by RTR. Note also the number of iterations needed to reach convergence with the RTR algorithm on the highly nonconvex `HUMPS` and `LOGHAIRY` problems is much higher than for the BTR algorithm. The complete numerical results are given in Section A.1 in appendices.

Our results show that the retrospective algorithm performs as well as the classical one and is just as reliable if the trust-region subproblem is solved approximately. However, if the problem size or structure allows an exact solution, the retrospective algorithm is then significantly more efficient (the improvement is typically of only a few iterations, but is very consistent) and just as reliable. A detailed analysis of our results shows that RTR is in general slightly more conservative than BTR in that it tends to take marginally shorter steps. However, this does not seem to alter performance in a negative way. In particular the longer steps of BTR often result in a larger proportion of unsuccessful iterations (this may be deduced from the result table since the number of unsuccessful iterations is given by the difference between the number of iterations and the number of gradient evaluations). We also note that the choice of an accurate minimization of Newton's model in the trust region also appears to be considerably more efficient than an approximate one, at least in terms of the number of iterations needed for convergence, irrespective of the choice between BTR and RTR. As a consequence, the retrospective variant is clearly at its best

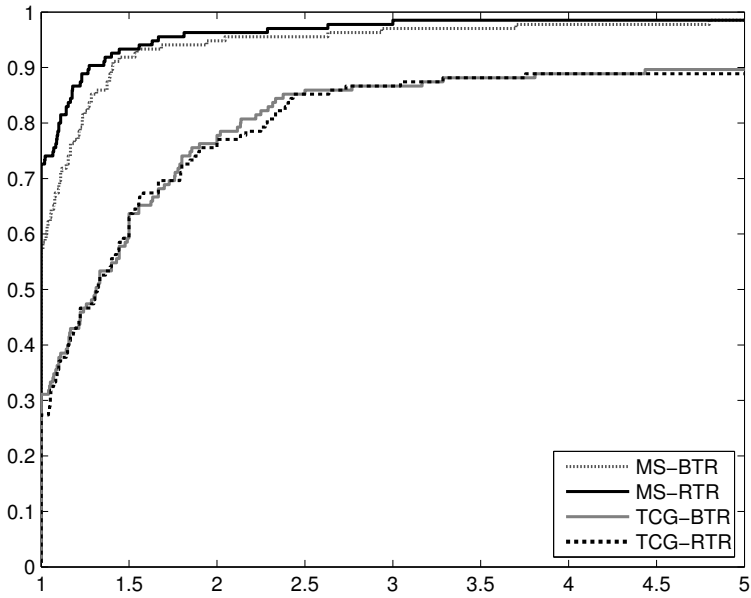


Figure 2.2 — *Performance profile comparing the number of iterations of the RTR and BTR algorithms.*

when the cost of evaluating the objective function and gradient dominates that of the overall iteration. Additional tests not reported here also indicate that both algorithms are essentially indistinguishable when quasi-Newton approximations (SR1 or BFGS, see Section 4.2) are used instead of the true Hessian. This is perhaps not surprising since the corresponding variants, which use exact solutions of approximate models, may also be interpreted as using approximate solutions of exact models.

2.8.4 Perspectives

This new method is especially interesting for adaptive techniques which exploit the information made available during the optimization process in order to vary the accuracy of the objective function computation. These methods typically appear in the context of a noisy objective function, where noise reduction can be achieved but at a significant cost. We therefore assume that the error can be estimated and consequently maintained under some acceptable threshold, while at the same time keeping the computational cost as low as possible. A first trust-region method with dynamic accuracy is described in Section 10.6 of Conn *et al.* (2000). The main idea there is to impose a model reduction larger than some multiple of the noise evaluated at both the current and candidate iterates. A cheaper nonmonotone approach has been developed in the context of nonlinear stochastic programming by Bastin, Cirillo and Toint (2006*a*) (see also Bastin, Cirillo and Toint, 2006*b*), more specifically for the minimization of sample average approximations (Shapiro, 2003) relying on Monte-Carlo sampling, a method also known as sample-path optimization (Robinson, 1996). The main difference with respect to the work of Conn *et al.* (2000) is that it allows a reduction of the model smaller than the noise level. In both cases, the size of the model reduction is the main component to decide on the desired accuracy of the objective function: the adaptive mechanism is thus applied on the basis of past information, at the previous iterate, rather that at the current one. Our new proposal could therefore improve these techniques significantly because it uses the most relevant information on the model's quality at the current iterate instead of at the previous iterate, but this remains to be analyzed further.

Chapter 3

Multilevel methods

In this chapter, we introduce the multilevel framework, by considering first the linear case in Section 3.1, and then the nonlinear one in Section 3.2, hence following its historical development. Whereas the word *multigrid* is often used in both contexts, we prefer to speak about *multilevel* methods in the nonlinear case.

Several developments in the next chapters are based on the multilevel elements that are described here. The RMTR method (Section 3.2) is indeed used for our numerical comparison of Hessian approximations for multilevel problems (Chapter 5) and for an application to snake-skin pigmentation patterns modelling (Chapter 7). Besides, the multilevel hierarchy is at the basis of the new limited-memory method introduced in Chapter 6.

3.1 Multigrid methods for linear systems

The solution of linear systems of equations

$$Ax = b \tag{3.1}$$

in the variable $x \in \mathbb{R}^n$, and where $A \in \mathbb{R}^{n \times n}$ and $b \in \mathbb{R}^n$, have interested mathematicians since antiquity (an empirical form of the Gaussian elimination was already known in China from the first century[1]) and have been especially a challenging field of research for nearly three centuries (see already Cramer's rule by Maclaurin, 1748, and Cramer, 1750). Linear systems often appear in applied mathematics, either because the modelled phenomenon has actually a linear behaviour, or simply because we are not able to deal directly with its nonlinearity. Think for instance at the methods presented in Chapter 2

[1] See the anonymous Chinese mathematics book *The Nine Chapters on the Mathematical Art*.

to solve nonlinear optimization problems: most of the time, they involve the (approximate) solution of a linear system. Besides, physical phenomena are often described in terms of partial differential equations. Their solution is a function, which generally means an infinite number of unknowns. To make the problem numerically tractable, we must then approximate the problem using only a finite number of variables, a step that is called *discretization*. This operation is typically achieved either with a *finite-difference approximation* (see our example in Subsection 3.1.1, as well as Strikwerda, 2004), or with a *finite-element method* (see among many others Ern and Guermond, 2004, and Gockenbach, 2006). The resulting problems usually involve a (very) large number of variables (to obtain an accurate representation of the true functional solution). Depending on whether the original problem is linear in the unknown function (and its derivatives), the discretized problem is either directly written as a linear system, or may be iteratively solved through a sequence of linear systems. In every case, we must be able to solve efficiently large linear systems, which are often fortunately *sparse*[2], in the sense that the matrix A has few nonzero entries (of the order of n, instead of n^2).

Methods designed to solve linear systems fall mostly in two classes: the *direct methods*, which use a factorization of the matrix A (see Chapters 3 and 4 in Golub and Van Loan, 1996), and the *iterative methods*, which build a sequence of iterates that hopefully converges to the problem solution (see Chapters 9 and 10 in the same reference). While direct methods may adapt to (sparse) large-scale problems (see for instance Davis, 2006), iterative methods are usually better suited to those problems (see Saad, 2003, for a comprehensive coverage of iterative methods). The latter methods may themselves be gathered in *Krylov subspace methods* (like the conjugate gradient method, see Subsection 2.6.1), and in *stationary linear methods*, which compute the solution by iteratively applying a constant affine operator to the current iterate until convergence. Amongst the latter, we present in Subsection 3.1.2 the still more particular subclass of *relaxation methods*. They are indeed involved in the latest revolution in this field: *multigrid methods*.

Although this property remained unexploited until the 1960s, many large-scale problems (and in particular, virtually all discretized problems) possess several descriptions at different levels of accuracy, what we call a *multilevel hierarchy* of descriptions. The idea of multigrid methods is therefore to exploit this structure while maintaining the global coherence of the solution process between the levels. Fedorenko (1964) first used this idea for the solution of the Poisson equation on the unit square (see our second toy problem in Subsection 3.1.1). However, the development of the multigrid methods only really starts with the pioneering work of Brandt (1973) who stated the main principles and practical utility of this framework. Hackbusch (1976) discovered indepen-

[2] By contrast, data assimilation problems (see Kalnay (2002) and Lewis, Lakshmivarahan and Dhall (2006), for an introduction) typically present a dense structure.

dently this concept and provided reliable multigrid methods (see Hackbusch, 1995). The end of this section is thus devoted to a brief description of the main principles involved in the design of multigrid methods (Subsection 3.1.3) and to two common resulting strategies: the multigrid correction (Subsection 3.1.4) and the mesh refinement (Subsection 3.1.5).

3.1.1 Toy problems

We introduce here two toy problems, formalized as linear systems, that will help us to illustrate further concepts in multilevel methods. First, the one-dimensional Poisson equation with Dirichlet boundary conditions:

$$\mathbf{u}'' = \mathbf{f} \quad \text{on } \Omega = (0,1), \tag{3.2a}$$

$$\mathbf{u} = 0 \quad \text{on } \partial\Omega = \{0,1\}. \tag{3.2b}$$

where the variable $\mathbf{u}$ is a sufficiently smooth function defined on Ω, and $\mathbf{f}$ is a given function also defined on Ω. This problem may be solved numerically using finite-difference approximation. In other words, the domain Ω is discretized (see Figure 3.1) by partitioning it into n subintervals defined by grid points $x_j = jh$ $(j = 0, \ldots, n)$ where $h = \frac{1}{n}$ is the mesh size, that is the length of each subinterval.

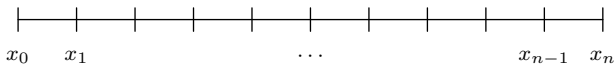


Figure 3.1 — *One-dimensional discretized domain* Ω_h.

We then approximate the function $\mathbf{u}$ by only considering its values u_j on the grid points x_j, yielding the new variable $u = (u_1, \ldots, u_{n-1})$ since the values u_0 and u_n are known to be zero due to (3.2b). This vector u must then satisfy the $n-1$ linear equations

$$\frac{-u_{j-1} + 2u_j - u_{j+1}}{h^2} = f_j \qquad (j = 1, \ldots, n-1), \tag{3.3}$$

where $f_j = \mathbf{f}(x_j)$ $(j = 1, \ldots, n-1)$ forms the components of a vector f of $\mathbb{R}^{n-1}$. The matrix representation of this linear system is

$$\frac{1}{h^2}\begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}\begin{pmatrix} u_1 \\ \\ \vdots \\ \\ u_{n-1} \end{pmatrix} = \begin{pmatrix} f_1 \\ \\ \vdots \\ \\ f_{n-1} \end{pmatrix} \tag{3.4}$$

or $Au = f$, where the matrix A is tridiagonal (so sparse), symmetric and positive definite.

Our second toy problem is the two-dimensional version of the former:

$$-\Delta \mathbf{u} = \mathbf{f} \quad \text{on } \Omega = (0,1) \times (0,1), \tag{3.5a}$$

$$\mathbf{u} = 0 \quad \text{on } \partial\Omega. \tag{3.5b}$$

where $\Delta \mathbf{u}$ denotes the Laplacian of u. In this case, the discretization is built on a grid Ω_h defined by the points $(x_i, y_j) = (ih_x, jh_y)$ where $h_x = \frac{1}{m}$ and $h_y = \frac{1}{n}$, as illustrated in Figure 3.2.

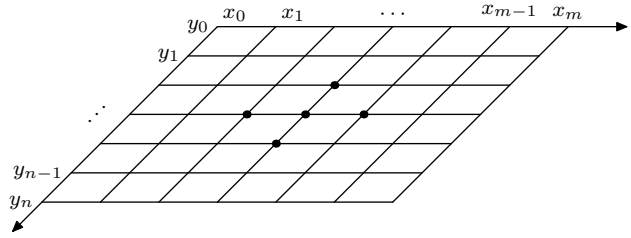


Figure 3.2 — *Two-dimensional discretized domain Ω_h. Solid dots indicate the unknowns related to a typical grid point.*

A finite-difference approximation of (3.5) then yields the equations

$$\frac{-u_{i-1,j} + 2u_{i,j} - u_{i+1,j}}{h_x^2} + \frac{-u_{i,j-1} + 2u_{i,j} - u_{i,j+1}}{h_y^2} = f_{i,j} \tag{3.6a}$$

$$u_{i,0} = u_{i,n} = u_{0,j} = u_{m,j} = 0, \tag{3.6b}$$

for $i = 1, \ldots, m-1$ and $j = 1, \ldots, n-1$, where $u_{i,j}$ is the approximated value of $\mathbf{u}$ at point (x_i, y_j), and where $f_{i,j} = \mathbf{f}(x_i, y_j)$.

In order to write the matrix representation of this linear system, we must first write the variable u as a vector of $\mathbb{R}^{(m-1)(n-1)}$ instead as an element of $\mathbb{R}^{(m-1)\times(n-1)}$. This means to define an order on the components $u_{i,j}$. On rectangular grids, the lexicographical order is a very common choice. It consists in numbering the variables $u_{i,j}$ row by row, hence placing the rows $u_i \overset{\text{def}}{=} (u_{i,1}, \ldots, u_{i,n-1})$ consecutively for i going from 1 to $n-1$. If we now define the $(n-1) \times (n-1)$ tridiagonal matrix

$$T = \begin{pmatrix} 2(h_x^{-2} + h_y^{-2}) & -h_y^{-2} & & & \\ -h_y^{-2} & 2(h_x^{-2} + h_y^{-2}) & -h_y^{-2} & & \\ & \ddots & \ddots & \ddots & \\ & & -h_y^{-2} & 2(h_x^{-2} + h_y^{-2}) & -h_y^{-2} \\ & & & -h_y^{-2} & 2(h_x^{-2} + h_y^{-2}) \end{pmatrix},$$

we can then write the equations (3.6) as

$$\begin{pmatrix} T & -h_x^{-2}I & & & \\ -h_x^{-2}I & T & -h_x^{-2}I & & \\ & \ddots & \ddots & \ddots & \\ & & -h_x^{-2}I & T & -h_x^{-2}I \\ & & & -h_x^{-2}I & T \end{pmatrix} \begin{pmatrix} u_1 \\ \\ \vdots \\ \\ u_{m-1} \end{pmatrix} = \begin{pmatrix} f_1 \\ \\ \vdots \\ \\ f_{m-1} \end{pmatrix} \tag{3.7}$$

where f_i is similarly defined as $(f_{i,1}, \ldots, f_{i,n-1})$ $(i = 1, \ldots, n-1)$.

3.1.2 Relaxation methods

Considering the solution x_* of the linear system (3.1), we first define the *error* on the solution at a given point x:

$$e \stackrel{\text{def}}{=} x_* - x, \tag{3.8}$$

which is *a priori* as inaccessible as the solution x_*. However, we can measure the quality of the approximation x, by computing the *residual*

$$r \stackrel{\text{def}}{=} b - Ax, \tag{3.9}$$

that is the amount by which the approximation x fails to satisfy the problem (3.1). The error is however not necessarily small when the residual is small. The error and residual are indeed linked by the relation

$$r = b - Ax = A(x_* - x) = Ae, \tag{3.10}$$

which is called the *residual equation*. The idea behind relaxation methods is then to build the new iterate x^+ in two stages: first the determination of an approximate solution e of the residual equation and then the *residual correction* $x^+ = x + e$. More formally, we compute

$$x^+ = x + Hr \tag{3.11}$$

for some matrix H approximating A^{-1}. Using (3.9), this step may be equivalently written as

$$x^+ = Rx + Hb \tag{3.12}$$

with $R = I - HA$, thus yielding a stationary linear method.

Jacobi iteration. — The first relaxation method was proposed by Jacobi. He decomposed the matrix A as

$$A = D - L - U, \tag{3.13}$$

where D is a diagonal matrix , L is a strictly lower-triangular matrix, and U is a strictly upper-triangular matrix. The Jacobi iteration is then

$$x^+ = R_J x + D^{-1}b \quad \text{with} \quad R_J = D^{-1}(L + U). \tag{3.14}$$

Gauss-Seidel iteration. — Using the same decomposition (3.13), Gauss and Seidel proposed an improvement of the Jacobi method:

$$x^+ = R_G x + (D - L)^{-1} b \quad \text{with} \quad R_G = (D - L)^{-1} U. \tag{3.15}$$

As the convergence speed of relaxation methods depends on the spectral radius $\rho(R)$ (see Subsection 10.1.2 in Golub and Van Loan, 1996), the Gauss-Seidel method typically converges faster than the Jacobi method. It also requires a little less memory, since the components of x^+ may be directly stored in place of that of x. Indeed, at a given iteration, Gauss-Seidel method updates the component x_i on the basis of the former updated components x_j^+ (for $j < i$), and of the remaining components x_j (for $j > i$). By contrast, Jacobi method uses always the current components x_j (for $j \neq i$). This different behaviour however allows the Jacobi method to be parallelized, but not the Gauss-Seidel method. It also implies that the Gauss-Seidel method is sensitive to the ordering of the variable components.

3.1.3 Smooth and oscillatory modes

The convergence of relaxation methods may be further analyzed using a discrete Fourier decomposition of the error. In the case of our first toy problem,

$$e = \sum_{\ell=1}^{n-1} c_\ell w_\ell \tag{3.16}$$

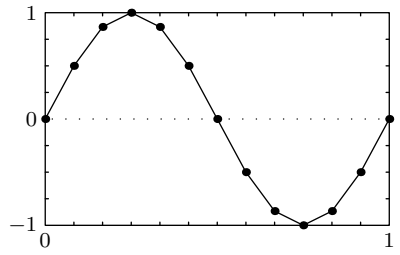
where the Fourier modes (or wave functions) w_ℓ are defined by

$$[w_\ell]_j = \sin\left(\frac{j\ell\pi}{n}\right) \tag{3.17}$$

for $j = 0, \ldots, n$ and $k = 1, \ldots, n-1$. Note that these vectors w_ℓ are the eigenvectors of the matrix A. The Fourier modes w_ℓ are *smooth* when $1 \leqslant \ell < \frac{n}{2}$, and *oscillatory* when $\frac{n}{2} \leqslant \ell \leqslant n$. This terminology is illustrated in Figure 3.3. Note that the limit frequency $\frac{n}{2}$ is called the *Nyquist frequency*.

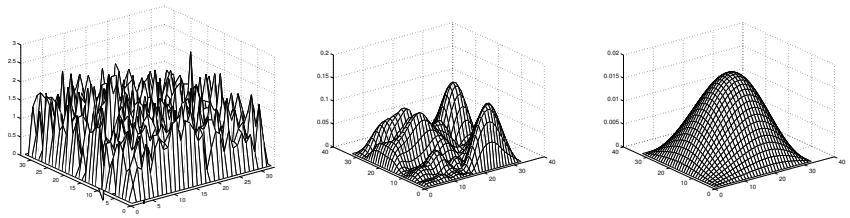
Smoothing principle. — Typically, relaxation methods effectively reduce the oscillatory modes of the error along the iterations, but have difficulties to reduce the smooth ones (see Trottenberg, Oosterlee and Schüller, 2001, or Wesseling, 1992). In Figure 3.4, we show in particular the effect of the Gauss-Seidel method for solving the discretized two-dimensional Poisson equation (3.6).

This *smoothing* property has the obvious drawback to significantly slow the convergence of relaxation methods if the initial error contains smooth modes (a fact that can not be prevented without *a priori* information on the solution). It is therefore the intent of multigrid methods to circumvent this drawback, by playing with the descriptions of the problems at different levels.



(a) w_{10}^h. (b) $-w_2^{2h}$.

Figure 3.3 — *A smooth mode* (a) *and an oscillatory mode* (b) *on* Ω_h *for* $n = 12$.



(a) *Initial error.* (b) *After 10 iterations.* (c) *After 100 iterations.*

Figure 3.4 — *Evolution of the error when the Gauss-Seidel method is applied to the 2-D Poisson problem. Note that the vertical scale is each time reduced.*

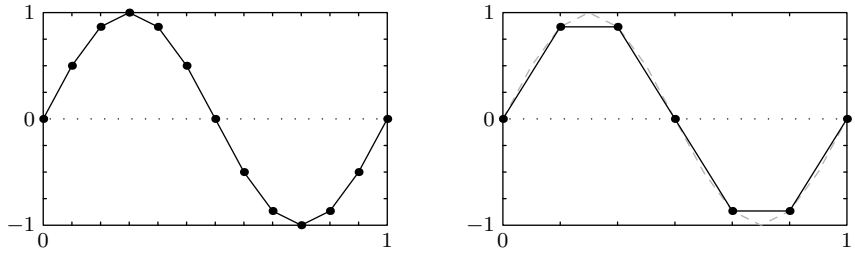
We now consider the application of relaxation methods to *multigrid problems*, which are problems that can be described at several levels as linear systems involving an increasing number of variables. Let us consider for simplicity a problem defined on two grids: a *fine* grid Ω_h with $n_h = \frac{1}{h}$ points, and a *coarse* grid Ω_{2h} with $n_{2h} = \frac{1}{2h} = \frac{n_h}{2}$ points. We use the superscript h ($2h$) to indicate that a quantity is defined on the grid Ω_h (Ω_{2h}). Choosing a mesh size ratio equal to 2 is a common practice since there is no advantage to take other ratios (see Briggs, Henson and McCormick, 2000). We make two observations on the representation of Fourier modes on these grids, and illustrate them graphically.

Coarse grid principle. — First, observe that Ω_{2h} consists in the even numbered points of the grid Ω_h. The *smooth* modes evaluated at these points are

$$[w_\ell^h]_{2j} = \sin\left(\frac{2j\ell\pi}{n_h}\right) = \sin\left(\frac{j\ell\pi}{n_{2h}}\right) = [w_\ell^{2h}]_j \tag{3.18}$$

for $1 \leqslant \ell < n_{2h}$ and $0 \leqslant j \leqslant n_{2h}$, that is all the Fourier modes defined on Ω_{2h}. This means that the smooth modes of Ω_h appear more oscillatory

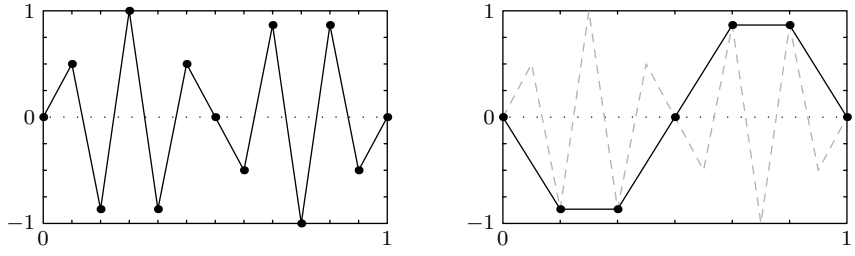
when represented on Ω_{2h} (see the example in Figure 3.5 on this page). We refer to this phenomenon as the *coarse grid principle*.



(a) w_2^h. (b) w_2^{2h}.

Figure 3.5 — *A smooth mode on* Ω_h (a) *and its representation on* Ω_{2h} (b) *(for* $n = 12$*). The dashed grey line in* (b) *is a copy of the mode illustrated in* (a).

Aliasing. — Second, a similar computation shows that the oscillatory mode $w^h_{n_h-\ell}$ (for $1 \leqslant \ell \leqslant n_{2h}$) of Ω_h is represented as $-w_\ell^{2h}$ on Ω_{2h}, (the opposite of) a Fourier mode that also represents the smooth mode w_ℓ^h. This phenomenon is called *aliasing*. Oscillatory modes of the fine grid are thus misrepresented on the coarse grid (see the example in Figure 3.6 on page 66). In fact, they can not be appropriately represented on the coarse grid due to Nyquist-Shannon sampling theorem (see for instance Luenberger, 2006).



(a) w_{10}^h. (b) $-w_2^{2h}$.

Figure 3.6 — *An oscillatory mode on* Ω_h (a) *and its representation on* Ω_{2h} (b) *(for* $n = 12$*). The dashed grey line in* (b) *is a copy of the mode illustrated in* (a).

3.1.4 Multigrid correction scheme

We now have the core elements that inspire the design of multilevel methods: the smoothing effect of relaxation methods, and the coarse grid principle.

Hence, we sketch the general and ideal behaviour of these methods. Starting on the fine grid, the oscillatory modes of the error are first removed with smoothing iterations (typically that of a relaxation method). The remaining smooth modes are then represented on the coarse grid, where they appear more oscillatory, and can thus be removed more efficiently by further smoothing iterations on the coarse grid. This second cycle is also cheaper since the number of variables is smaller on the coarse grid than on the fine one.

Before describing this process in more detail, we need to formalize how information can be transferred from one grid to the other. We thus assume that two linear operators are given for that purpose: a *prolongation operator* $P^h : \mathbb{R}^{n_{2h}} \to \mathbb{R}^{n_h}$ (also often called the interpolation operator) and a *restriction* operator $R^h : \mathbb{R}^{n_h} \to \mathbb{R}^{n_{2h}}$.

The *multigrid correction* scheme formalizes our former sketch and is the basic operation in multigrid methods. First, a cycle of ν_1 smoothing iterations is applied on the fine grid to eliminate most of the oscillatory modes of the error e^h at that level. We then pursue on the coarse level. There, following the philosophy of relaxation methods, we should solve a coarse version of the residual equation $A^h e^h = r^h$. Although we could obtain it from the problem description on the coarse grid, it is more common to use instead the *Galerkin operator*:

$$A^{2h} = R^h A^h P^h \tag{3.19}$$

(see Briggs *et al.*, 2000, for a justification of this choice), and the restricted residual $r^{2h} = R^h r^h$. residual equation $A^{2h} e^{2h} = r^{2h}$ hence gives the coarse error e^{2h}, which is then projected on the fine grid using the prolongation operator: $e^h = P^h e^{2h}$, yielding a residual correction. This operation typically reintroduces oscillatory modes a little, which are then reduced with ν_2 additional smoothing iterations on the fine grid. This yields Algorithm 3.1.

Algorithm 3.1 (Multigrid correction scheme)

An iterate x^h on the fine grid and two nonnegative integer constants ν_1 and ν_2 are given.

Step 1. Presmoothing: Apply ν_1 iterations of a smoothing procedure on x^h.

Step 2. Coarse-grid correction:

(a) Compute the fine residual $r^h = b^h - A^h x^h$, its restriction $r^{2h} = R^h r^h$, and the Galerkin operator $A^{2h} = R^h A^h P^h$.

(b) Solve the coarse residual equation $A^{2h} e^{2h} = r^{2h}$ for e^{2h}.

(c) Prolongate the coarse correction: $e^h = P^h e^{2h}$, and use it to correct the fine iterate: $x^h \leftarrow x^h + e^h$.

Step 3. Postsmoothing: Apply ν_2 additional iterations of a smoothing procedure on x^h.

This procedure may be applied recursively if more than two levels are at our disposal. The residual equation at Step 2 (b) is then solved itself using the multigrid correction scheme, and so on. At the coarsest level, the residual equation may be solved in contrast by a direct method since the problem size is then small.

Commonly, one or two multigrid corrections are performed to solve each residual equation (except for the coarsest one, which is solved in a classical way), yielding the V-cycle (Figure 3.7) and W-cycle (Figure 3.8) schemes, respectively, named after the shape of the iterations representations.

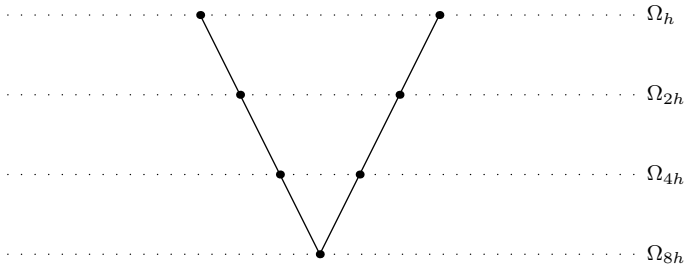


Figure 3.7 — *The V-cycle scheme on four levels.*

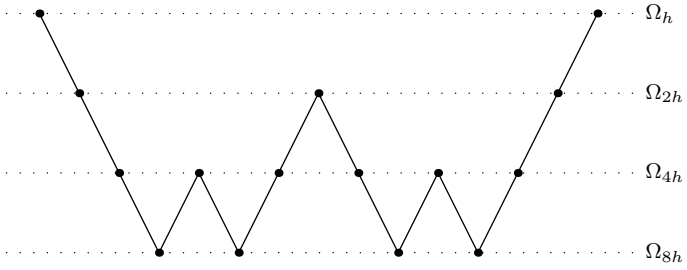


Figure 3.8 — *The W-cycle scheme on four levels.*

3.1.5 Mesh refinement

Another strategy that multigrid problems enable is called *mesh refinement*. It consists in using the coarser levels to define a good starting point on the fine level. Starting from the coarsest level, the problem is thus solved at the current

level, and the resulting solution is prolongated on the finer level to serve as the starting point of the solving process at that level.

This simple strategy brings an important improvement of the convergence speed of the global process. Indeed, a good starting point usually helps significantly iterative methods to converge faster, and on the other hand, the additional cost to compute this starting point (using the coarser levels) is strongly mitigated by the cheaper cost of computations on the coarse grids with respect to that on the fine grid.

This technique may be combined with the multigrid correction techniques. For instance, the *full multigrid* (FMG) scheme performs a V-cycle scheme to compute the problem solution at each of the increasingly finer grids used in the mesh refinement, as illustrated in Figure 3.9.

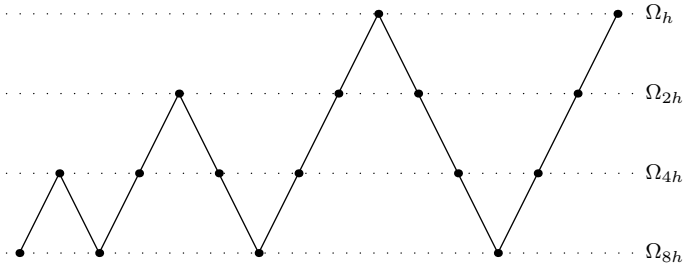


Figure 3.9 — *The full multigrid scheme on four levels.*

More information on multigrid methods may be found in the tutorial book by Briggs *et al.* (2000), in Mouffe (2005), and finally in Trottenberg *et al.* (2001) for a comprehensive coverage.

3.2 Recursive multilevel trust-region method

We turn back to nonlinear optimization, after having outlined the basic principles of multigrid methods for linear systems. We consider here the (possibly constrained) optimization problem of the form

$$\min_{x \in \mathcal{F}} f(x), \tag{3.20}$$

where f is a twice-continuously differentiable objective function which maps $\mathbb{R}^n$ into $\mathbb{R}$ and is bounded below. We also assume that an appropriate hierarchy of descriptions is known for the problem under consideration. To be more specific, suppose that a collection of functions $\{f_i\}_{i=0}^r$ is available, where each function f_i is twice-continuously differentiable and maps $\mathbb{R}^{n_i}$ into $\mathbb{R}$, and where $n_r = n$ and $f_r = f$, giving back our original problem. To fix terminology, we will refer to a particular i as a level and say that level p is coarser than level q if $p < q$ and that level p is finer than level q if $p > q$. We assume

that this multilevel hierarchy is appropriate in the sense that each function f_i still describes the problem (3.20) reasonably well and is cheaper to minimize than its finer version f_{i+1}. This type of multilevel structure arises in a variety of forms and applications, but the most common is probably again the discretized infinite-dimensional framework where the functions f_i represent increasingly finer discretizations of the same infinite-dimensional problem. Such a multilevel hierarchy can therefore be used to improve the efficiency of the numerical solution on the finest level, similarly to the seminal work on linear systems.

The numerical exploitation of multilevel structure in nonlinear optimization has been the object of several contributions in the past few years. Surface design, data assimilation for weather forecasting (Fisher, 1998) and optimal control of systems described by partial-differential equations have been the main motivation of this challenging research trend, but other applications such as multi-dimensional scaling (Bronstein, Bronstein, Kimmel and Yavneh, 2005), progressive lens design (see Toint and Tomanos, 2009, and Section 4.4 of Tomanos, 2009) are also of interest. Several algorithms have been proposed to exploit the multilevel structure in the optimization context. Successful multilevel linesearch methods have been proposed by Fisher (1998), Nash (2000*a*), Lewis and Nash (2005), Oh, Milstein, Bouman and Webb (2003), Wen and Goldfarb (2007) and Gratton and Toint (2010) (see Chapter 6). The rest of this section is however devoted to the presentation of a multilevel method of the trust-region type: the *Recursive Multilevel Trust-Region* (RMTR) method introduced by Gratton, Sartenaer and Toint (2008*b*) suitable for unconstrained optimization (that is $\mathcal{F} = \mathbb{R}^n$) and using the Euclidean norm to define the trust-region. Gratton, Mouffe, Toint and Weber-Mendonça (2008*a*) design afterwards an improved version using the infinity norm, and which is then able to also tackle bound-constrained optimization problems. We focus on the description of the later variant, which has a broader scope of application and results in substantial algorithmic simplifications when compared to the earlier algorithm (we refer the reader to Gratton *et al.*, 2008*a*, for a more complete discussion of these advantages). Gratton, Mouffe, Sartenaer, Toint and Tomanos (2010*b*) have then discussed extensive numerical experiments with this method, and have compared it with other algorithms on a library of discretized infinite-dimensional problems. The obtained results are excellent and suggest that the method can be of interest more widely.

3.2.1 Description

We thus assume that the set $\mathcal{F}$ is defined as $\{x \in \mathbb{R}^n | \, l \leqslant x \leqslant u\}$, where some components of the bound constraints l and u are possibly infinite. To complete the multilevel hierarchy of descriptions of the problem, some relation must exists between the variables of consecutive levels. We hence assume that, for each level $i = 1, \ldots, r$, there exist a linear full-rank *restriction* operator R_i

from $\mathbb{R}^{n_i}$ to $\mathbb{R}^{n_{i-1}}$ and a linear full-rank *prolongation* operator P_i from $\mathbb{R}^{n_{i-1}}$ to $\mathbb{R}^{n_i}$ such that

$$P_i = \sigma_i R_i^T, \tag{3.21}$$

for some constant $\sigma_i > 0$ (see again Briggs *et al.*, 2000, for an excellent introduction, or Trottenberg *et al.*, 2001, for a more extensive coverage). We also assume that the restriction operator is normalized to ensure $\|R_i\|_\infty = 1$, and also that the entries of P_i and R_i are nonnegative.

Models definition. — Being based on the trust-region framework (see Section 2.7), the RMTR method computes its steps by (approximately) minimizing some models inside some trust regions. It however uses a two-pronged strategy to define the model at a given iteration k. First, it may use (as many practical trust-region algorithms) the quasi-Newton quadratic model

$$m_k(x_k + s) = f(x_k) + \langle g_k, s\rangle + \tfrac{1}{2}\langle s, B_k s\rangle, \tag{3.22}$$

where $B_k \in \mathbb{R}^{n\times n}$ is a symmetric approximation of $\nabla^2 f(x_k)$. A sufficient decrease in this model inside the trust region is then obtained by (approximately) solving the problem

$$\min_{\substack{\|s\|_\infty \leqslant \Delta_k \\ x_k + s \in \mathcal{F}}} m_k(x_k + s), \tag{3.23}$$

given some trust-region radius Δ_k. The choice of the infinity norm in the trust-region description is natural in the context of bound-constrained problems, because the feasible set for problem (3.23) can then be fully represented by bound constraints.

The philosophy of the RMTR method is however to use the multilevel hierarchy to efficiently construct minimization steps. This yields the second strategy to compute an appropriate step. More precisely, considering $x_{i,k}$, the k-th iterate at level $i > 0$, we first build a local coarse model $h_{i-1}(x_{i-1,0} + s_{i-1})$ around the restricted point $x_{i-1,0} = R_i x_{i,k}$. We then minimize this model (using a trust-region method) inside a coarse set of bound constraints $\mathcal{L}_i$, which represent both the feasibility with respect to the original problem bound constraints, and the constraints on the step size inherited from the trust regions of the finer levels. Let $x_{i-1,*}$ thus be the (approximate) solution of this local coarse subproblem at level $i-1$, given by

$$\min_{x_{i-1,0} + s_{i-1} \in \mathcal{L}_{i-1}} h_{i-1}(x_{i-1,0} + s_{i-1}). \tag{3.24}$$

The coarse move $s_{i-1} = x_{i-1,*} - x_{i-1,0}$ is finally prolongated (using P_i) into a trust-region step s_i at level i. The details for the construction of the coarse set of bound constraints $\mathcal{L}_i$ are beyond the scope of this presentation, but the interested reader may find their full description in Gratton *et al.* (2008*a*), in Mouffe (2009), and in Weber Mendonça (2009).

Model selection. — However, the decision to use h_{i-1} as a model for f_i is not always the best, and the algorithm has a mechanism to decide whether using the local coarser-level model is useful. This decision is made by comparing criticality measures at the current and coarser levels. Let first $g_{i,k}$, denote the gradient of the function h_i at $x_{i,k}$. We then define the criticality measure at level i as

$$\chi_{i,k} \stackrel{\text{def}}{=} \chi(x_{i,k}) = \Big| \min_{\substack{x_{i,k}+d\in\mathcal{L}_i \\ \|d\|_\infty \leqslant 1}} \langle g_{i,k}, d\rangle \Big|, \tag{3.25}$$

which can be interpreted as the maximal decrease of the linearized problem that can be achieved in the intersection of $\mathcal{L}_i$ and a box of radius one (see notably Conn, Gould, Sartenaer and Toint, 1993). We then consider that recurring to the coarser level is useful whenever this decrease at level $i-1$ is significant compared to that achievable at level i, which we formalize by the condition that

$$\sigma_i \chi_{i-1,0} \geqslant \kappa_\chi \chi_{i,k}, \tag{3.26}$$

for some constant $\kappa_\chi \in (0,1)$. If (3.26) does not hold, then the algorithm resorts to using the quadratic model (3.22) at level i, which we denote by $m_{i,k}(x_{i,k}+s_i)$. Otherwise, the choice between the two models remains open, allowing the efficient exploitation of multigrid techniques such as smoothing iterations (see Subsection 3.2.4 below).

An outline of the algorithm. — We are now ready to outline the RMTR algorithm using the ideas developed above. This outline is presented as Algorithm 3.2.

Algorithm 3.2 (Recursive multilevel trust-region method)

$$\mathbf{RMTR}_\infty(i, x_{i,0}, g_{i,0}, \chi_{i,0}, \mathcal{L}_i, \epsilon_i^\chi)$$

The level index i $(0 \leqslant i \leqslant r)$, a starting point $x_{i,0}$, the gradient $g_{i,0}$ and criticality measure $\chi_{i,0}$ at that point, the criticality threshold ϵ_i^χ, the initial trust-region radius $\Delta_{i,0}$, and the bound-constraints set $\mathcal{L}_i$ are given.

Step 0. Initialization: Compute $f_i(x_{i,0})$. Set $k=0$ and

$$\mathcal{W}_{i,0} = \mathcal{L}_i \cap \mathcal{B}_{i,0}, \tag{3.27}$$

where $\mathcal{B}_{i,0} = \{x_{i,0}+s_i \in \mathbb{R}^{n_i} : \|s_i\|_\infty \leqslant \Delta_{i,0}\}$.

Step 1. Model choice: If $i=0$, go to Step 3. Else, compute $R_i x_{i,k}$, $R_i g_{i,k}$, $\mathcal{L}_{i-1}$ and $\chi_{i-1,0}$. If (3.26) fails, go to Step 3. Otherwise, choose to go to Step 2 or to Step 3.

Step 2. Recursive step computation: Call recursively Algorithm 3.2:

$$\mathbf{RMTR}_{\infty}(i-1, R_i x_{i,k}, R_i g_{i,k}, \chi_{i-1,0}, \mathcal{L}_{i-1}, \epsilon_{i-1}^{\chi}),$$

yielding an approximate solution $x_{i-1,*}$ of (3.24). Then define $s_{i,k} = P_i(x_{i-1,*} - R_i x_{i,k})$, set $\delta_{i,k} = \sigma_i[h_{i-1}(R_i x_{i,k}) - h_{i-1}(x_{i-1,*})]$ and go to Step 4.

Step 3. Taylor step computation: Choose $B_{i,k}$ and compute a step $s_{i,k} \in \mathbb{R}^{n_i}$ that sufficiently reduces the model

$$m_{i,k}(x_{i,k} + s_i) = h_i(x_{i,k}) + \langle g_{i,k}, s_i \rangle + \tfrac{1}{2} \langle s_i, B_{i,k} s_i \rangle \qquad (3.28)$$

and such that $x_{i,k} + s_{i,k} \in \mathcal{W}_{i,k}$. Set $\delta_{i,k} = m_{i,k}(x_{i,k}) - m_{i,k}(x_{i,k} + s_{i,k})$.

Step 4. Acceptance of the trial point: Compute $f_i(x_{i,k} + s_{i,k})$ and

$$\rho_{i,k} = \big[f_i(x_{i,k}) - f_i(x_{i,k} + s_{i,k})\big]/\delta_{i,k}. \qquad (3.29)$$

If $\rho_{i,k} \geqslant \eta_1$, then define $x_{i,k+1} = x_{i,k} + s_{i,k}$; otherwise, define $x_{i,k+1} = x_{i,k}$.

Step 5. Termination: Compute $g_{i,k+1}$ and $\chi_{i,k+1}$. If $\chi_{i,k+1} \leqslant \epsilon_i^{\chi}$ or $x_{i,k+1}$ is not feasible with respect to the bound constraints inherited from the upper levels, then return with the approximate solution $x_{i,*} = x_{i,k+1}$.

Step 6. Trust-Region update: Set

$$\Delta_{i,k+1} \in \begin{cases} [\Delta_{i,k}, +\infty) & \text{if } \rho_{i,k} \geqslant \eta_2, \\ [\gamma_2 \Delta_{i,k}, \Delta_{i,k}] & \text{if } \rho_{i,k} \in [\eta_1, \eta_2), \\ [\gamma_1 \Delta_{i,k}, \gamma_2 \Delta_{i,k}] & \text{if } \rho_{i,k} < \eta_1, \end{cases} \qquad (3.30)$$

and $\mathcal{W}_{i,k+1} = \mathcal{L}_i \cap \mathcal{B}_{i,k+1}$ where

$$\mathcal{B}_{i,k+1} = \{x_{i,k+1} + s_i \in \mathbb{R}^{n_i} : \|s_i\|_{\infty} \leqslant \Delta_{i,k+1}\}.$$

Increment k by one and go to Step 1.

3.2.2 Convergence properties

We now present the main convergence results of this method as established by Gratton *et al.* (2008*a*) (see also Mouffe, 2009, and Weber Mendonça, 2009).

As is common for trust-region methods (see assumptions A.1 and A.3, on page 45), we assume that the Hessians of the functions f_i and quadratic models $m_{i,k}$ are bounded above along the iterations. We also assume that all gradients (at all levels) remain uniformly bounded, *i.e.* there exists $\kappa_{\mathrm{g}} \geqslant 1$ such that

$$\|\nabla h_i(x_i)\| \leqslant \kappa_{\mathrm{g}} \quad \text{for all } i = 0, \ldots, r \text{ and all } x_i \in \mathcal{F}_i \tag{3.31}$$

where $\mathcal{F}_i$ is the restriction of $\mathcal{F}$ at level i. This assumption is not overly restrictive and, for instance, automatically holds by continuity if all iterates remain in a bounded domain, which is the case if both l and u are finite.

If Step 2 is taken at iteration (i, k), this iteration initiates a *minimization sequence* at level $i - 1$, which consists of all successive iterations *at this level* (starting from the point $x_{i-1,0} = R_i x_{i,k}$) until a return is made to level i within iteration (i, k). Moreover, for each iteration (i, k), we define the set $\mathcal{R}(i, k)$ containing the iterations (j, ℓ) that occur within iteration (i, k). Figure 3.10 illustrates these concepts.

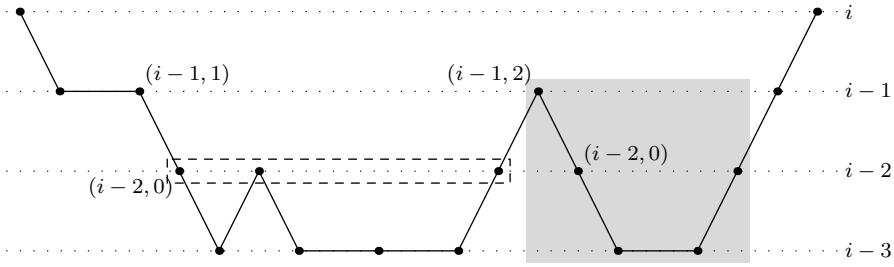


Figure 3.10 — *Illustration of some multilevel notations: the dashed rectangle area contains a minimization sequence at level $i-2$ initiated at iteration $(i-1,1)$, and the shaded rectangle contains $\mathcal{R}(i-1,2)$.*

The following result indicates that every recursive step at level i is potentially useful, in the sense that the coarse move $x_{i-1,\ell} - x_{i-1,0}$ becomes eventually non zero (that is for ℓ large enough). This also allows the use of multigrid schemes as V- and W-cycles, in which one or two successful iterations, respectively, are performed in every minimization sequence.

Theorem 3.1 (Gratton *et al.*, 2008*a*, Lemma 4.7) *Each minimization sequence contains at least one successful iteration.*

We next show the crucial result that the algorithm is well defined, and that all the recursions are finite. A bound on the maximum number of iterations may then be proved but is not presented here (see the aforementioned references).

Theorem 3.2 (Gratton *et al.*, 2008*a*, Theorem 4.9) *The number of iterations in each level is finite. Moreover, there exists* $\kappa_{\rm h} \in (0,1)$ *such that, for every minimization sequence at level* $i = 0, \ldots, r$ *and every* $t \geqslant 0$,

$$h_i(x_{i,0}) - h_i(x_{i,t+1}) \geqslant \tau_{i,t}\mu^{i+1}\kappa_{\rm h},$$

where $\tau_{i,t}$ *is the total number of successful Taylor iterations in* $\bigcup_{\ell=0}^{t} \mathcal{R}(i,\ell)$, *and* $\mu = \eta_1/\sigma_{\max}$ *with* $\sigma_{\max} = \max(1, \max_{i=1,\ldots,r} \sigma_i)$.

As a consequence, the RMTR algorithm is shown to be globally convergent, in the sense that, if ϵ_r is "driven to zero", it generates a sequence of iterates that are asymptotically first-order critical. More specifically, we examine the sequence of iterates $\{x_{r,k}\}$ generated as follows: we consider, at level r, a sequence of tolerances $\{\epsilon_{r,j}\} \subset (0,1)$ monotonically converging to zero, then start the algorithm with $\epsilon_r = \epsilon_{r,0}$ and alter slightly the mechanism of Step 5 (at level r only) to reduce ϵ_r from $\epsilon_{r,j}$ to $\epsilon_{r,j+1}$ as soon as $\chi_{r,k+1} \leqslant \epsilon_{r,j}$. The calculation is then continued with this more stringent threshold until it is also attained, ϵ_r is then again reduced and so on.

Theorem 3.3 (Gratton *et al.*, 2008*a*, Theorem 4.17) *Assume that* ϵ_r *is "driven to zero" in Algorithm 3.2. Then*

$$\lim_{k\to\infty} \chi_{r,k} = 0$$

3.2.3 Test problems

To test the numerical performance of the RMTR algorithm, multilevel problems have been gathered by Gratton *et al.* (2008*b*) and Gratton *et al.* (2010*b*). These problems are posed in functional spaces and involve differential operators. In Table 3.1, we give a brief description of each one. We however refer to Gratton *et al.* (2010*b*) and Tomanos (2009) for a detailed description. Note also that several of them come from the MINPACK-2 library by Averick, Carter, Moré and Xue (1992).

3.2.4 Numerical experiments

After a broad discussion on the value of the parameters in the RMTR method (see also Tomanos, 2009), Gratton *et al.* (2010*b*) present numerical experiments showing the good behaviour of the RMTR method. Before reporting them, we first need to detail how the choice in Step 1 was made in their numerical

Name	C	M	Type	Short description
DNT			1-D, quadratic	Dirichlet to Neumann transfer problem
P2D			2-D, quadratic	Poisson model problem
P3D			3-D, quadratic	Poisson model problem
DEPT		*	2-D, quadratic	Elastic-plastic torsion problem
DPJB	*	*	2-D, quadratic	Journal bearing problem
DODC		*	2-D, convex	Optimal design (composite materials)
MINS-SB			2-D, convex	Minimal surface (smooth boundary)
MINS-OB			2-D, convex	Minimal surface (oscillatory boundary)
MINS-BC	*		2-D, convex	Minimal surface
MINS-DMSA		*	2-D, convex	Minimal surface
IGNISC			2-D, convex	Combustion problem
DSSC		*	2-D, convex	Combustion problem
BRATU			2-D, convex	Combustion problem
MEMBR	*		2-D, convex	Membrane problem (free boundary)
NCCS			2-D, nonconvex	Optimal control (smooth boundary)
NCCO			2-D, nonconvex	Optimal control (oscillatory boundary)
MOREBV			2-D, nonconvex	Boundary value problem

Table 3.1 — *Our set of multilevel test problems. A star in the column* C *indicates that the problem has bound constraints, while one in the column* M *indicates that it comes from the MINPACK-2 library.*

implementation, and how the coarse model was defined and minimized inside the bound constraints. This implementation is available as the RMTR package inside the GALAHAD library of Gould, Orban and Toint (2003*b*), and we refer to the package documentation (Appendix D) for further details.

Recursion patterns and model selection. — Note that at the coarsest level ($i = 0$), no further recursion is possible. In this case, a Taylor's (that is non-recursive) iteration is the only possibility. At other levels, we must choose between the coarse model h_{i-1} and the Taylor's model $m_{i,k}$ when (3.26) holds. We now explain the two alternatives proposed by the RMTR package to perform this choice.

The first possibility consists to never use the coarse model. In this case, we may still use the multilevel hierarchy as mentioned in Subsection 3.1.5 to start from a good initial point (*mesh refinement* strategy, MR), or stay always on the finest level, leading to a standard trust-region algorithm (*all on finest* strategy, AF).

In the second possibility, the algorithm normally uses the coarse model each time this is allowed by (3.26). It may nevertheless ensure that *recursive iterations* (at which the coarse model is used) are preceded and/or followed by a *Taylor's iteration* (at which the Taylor's model is used), like the pre- and post-smoothing in the Algorithm 3.1. By default in the RMTR package,

one (smoothing) Taylor's iteration is performed before and after each recursive iteration. Again, the choice to use the coarse model may be associated with that to start from a good initial point as described above, leading to the *full multilevel* (FM) strategy (so called by analogy with the full multigrid scheme for linear systems). Applying directly the algorithm on the finest level defines on the contrary the *multilevel on finest* (MF) strategy.

Local coarse model. — Although other strategies are possible, the RMTR package use by default the *Galerkin approximation*

$$h_{i-1}(x_{i-1,0} + s_{i-1}) = \langle R_i g_{i,k}, s_{i-1} \rangle + \frac{1}{2} \left\langle s_{i-1}, R_i \nabla^2 h_i(x_{i,k}) P_i s_{i-1} \right\rangle \quad (3.32)$$

to define its local coarse model. This choice interestingly requires no evaluation of f_{i-1} or its derivatives, and is covered by the theory presented in Gratton *et al.* (2008*a*). Moreover, Gratton *et al.* (2010*b*) report better performance compared to other tested models.

Solving the subproblem. — Except at the coarsest level where the subproblem is solved with a projected truncated conjugate gradient method (PTCG method; see Conn, Gould and Toint (1988*a*, 1992), Tomanos (2009)), the subproblem is solved with a method inspired by the Gauss-Seidel method. Indeed, the model is successively minimized along the coordinate axis, yielding the Sequential Coordinate Minimization (SCM; see Ortega and Rheinboldt (1970), Section 14.6). This process has been shown to act as a smoothing procedure.

RMTR performance. — A comparison of these four strategies (AF, FM, MF and MR) was conducted by Gratton *et al.* (2010*b*) on the problems presented in Subsection 3.2.3. The performance profile in Figure 3.11 show that the *full multilevel* (FM) strategy clearly outperforms all other strategies. The second observation is that the *all on finest* (AF) strategy is, as expected, by far the worst. The remaining two strategies (MF and MR) are surprisingly close, and the use of recursive iterations on the fine level appears to have an efficiency similar to that of optimizing on successively finer grids.

The complete numerical results and their detailed analysis can be found in Gratton *et al.* (2010*b*) and Tomanos (2009).

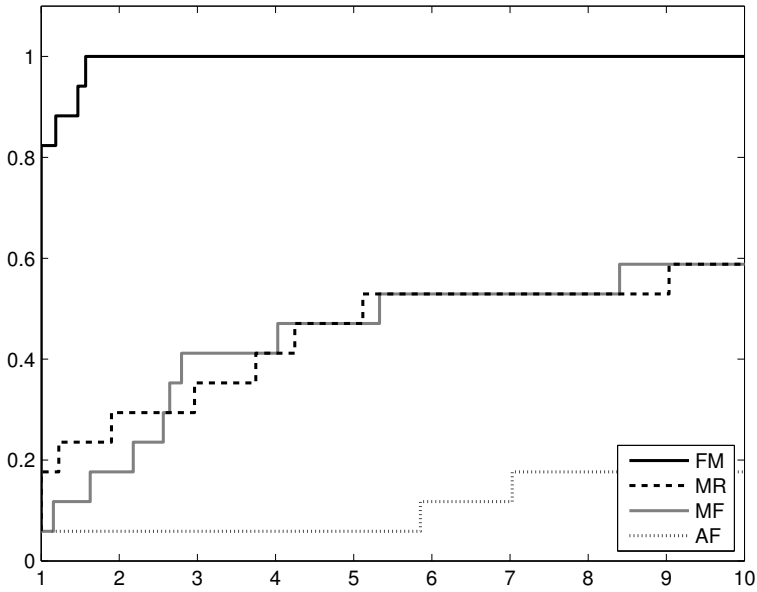


Figure 3.11 — *Performance profile for the RMTR method, comparing the* AF, FM, MF *and* MR *variants with the CPU time as criterion.*

Part II

Hessian approximation

Chapter 4

Introduction to Hessian approximation

Derivatives may considerably improve the performance of optimization methods. Compare for instance the linear rate of convergence of the steepest descent method, which does not use second-order information, with the quadratic rate of Newton's method, which uses the (exact) Hessian matrix. However, computing this matrix may be a tough task, notably because of its computation time cost and/or because of its memory requirements. Numerical procedures providing surrogate information on the second derivatives fall into three main classes: first the finite-difference methods (which use gradient differences), then the secant methods (which are based on more specific secant equations), and finally the automatic differentiation methods (which compute the derivatives given an explicit analytical expression of the function). It is the purpose of this chapter to briefly introduce these classes of methods.

4.1 Finite-difference methods

4.1.1 Gradient approximation

Given a function $f : \mathbb{R}^n \to \mathbb{R}$, finite-difference methods basically rely on the definition of derivatives (1.38) to (approximately) compute the gradient of f at some point $x \in \mathbb{R}$, yielding the *forward finite-difference approximation*:

$$[\bar{g}_h(x)]_i = \frac{f(x + h_i e_i) - f(x)}{h_i} \qquad (i = 1, \ldots, n), \tag{4.1}$$

for some vector $h \in \mathbb{R}^n$ of difference step lengths. This computation requires n function evaluations, if we assume that $f(x)$ has already been computed.

Note that if the function f is uniformly bounded (as stated in assumption A.1, page 45), then the following bound (see Subsection 8.4.3 of Conn *et al.*, 2000) holds:

$$\|\nabla f(x) - \bar{g}_h(x)\| \leqslant \tfrac{1}{2}\kappa_{\text{ufh}}\|h\|. \tag{4.2}$$

Alternatively, we may use the second-order accurate *central finite-difference approximation*:

$$[\hat{g}_h(x)]_i = \frac{f(x + h_i e_i) - f(x - h_i e_i)}{2h_i} \qquad (i = 1, \ldots, n), \tag{4.3}$$

for which we have the theoretical bound

$$\|\nabla f(x) - \hat{g}_h(x)\| \leqslant \tfrac{1}{6}\kappa_{\text{ufh}}\|h\|^2 \tag{4.4}$$

(see the same reference as above). This improvement in terms of accuracy yet doubles the number of function evaluations.

A fundamental question herein is which values we should use for the difference step lengths h, since the concept of number tending to zero has no sense in finite precision arithmetic. Indeed, if h is (relatively) smaller than the machine precision ϵ_{M}, then the points x and $x + h_i e_i$ are numerically indistinguishable and the derivative appears to be zero for any function. Moreover, even if h is a little larger than ϵ_{M}, numerical errors may prevail in the computation of the function values difference $f(x + h_i e_i) - f(x)$ and still be accentuated afterwards with the division by h_i. On the other hand, too large values of h no more guarantee a good approximation of the derivative (as stated by (4.2) and (4.4)). While this question is problem dependent, it is commonly advised to choose

$$h_i = \text{sign}(x_i)\sqrt{\epsilon_{\text{M}}}\max(|x_i|, S_{ii}) \tag{4.5}$$

for forward finite-difference approximations, and

$$h_i = \text{sign}(x_i)\sqrt[3]{\epsilon_{\text{M}}}\max(|x_i|, S_{ii}) \tag{4.6}$$

for central finite-difference approximations, where S is a diagonal scaling matrix. Considering for instance the case where double precision is used ($\epsilon_{\text{M}} = 2^{-53}$) and $x_i = S_{ii} = 1$, the advised value of h_i is thus approximately $1.1 \cdot 10^{-8}$ in the forward case, and $4.8 \cdot 10^{-6}$ in the central case. Note that central finite-difference approximation are typically only used when the gradient becomes small (see for instance Stewart, 1967).

4.1.2 Hessian approximation

Hessians can be computed with finite differences in two ways, either from function evaluations or from gradient evaluations. In the first case, we apply twice the forward approximation (4.1) to obtain the Hessian approximation

$$[\check{B}_h(x)]_{ij} = \frac{f(x + h_i e_i + h_j e_j) - f(x + h_i e_i) - f(x + h_j e_j) + f(x)}{h_i h_j} \tag{4.7}$$

for $i, j = 1, \ldots, n$. This operations is highly expensive since, even taking into account the symmetry, it requires $n(n+1)/2$ additional function evaluations to that of $\bar{g}_h(x)$. The recommended difference step length is then given by

$$h_i = \text{sign}(x_i)\sqrt[4]{\epsilon_{\text{M}}}\max(|x_i|, S_{ii}). \tag{4.8}$$

Considering again the case where double precision is used and $x_i = S_{ii} = 1$, the advised value of h_i is thus approximately $1.0 \cdot 10^{-4}$.

In the second case, forward/central finite-difference approximation is applied on each component of the gradient yielding the approximations

$$[\bar{B}_h^*(x)]_{ij} = \frac{[\nabla f(x + h_i e_i) - \nabla f(x)]_j}{h_i} \qquad (i, j = 1, \ldots, n), \tag{4.9}$$

in the forward case, and

$$[\hat{B}_h^*(x)]_{ij} = \frac{[\nabla f(x + h_i e_i) - \nabla f(x - h_i e_i)]_j}{2h_i} \qquad (i, j = 1, \ldots, n), \tag{4.10}$$

in the central case. To ensure the symmetry of the generated approximation, we however prefer the symmetrized version

$$\bar{B}_h(x) = \tfrac{1}{2}[\bar{B}_h^*(x) + \bar{B}_h^*(x)^T] \text{ and } \hat{B}_h(x) = \tfrac{1}{2}[\hat{B}_h^*(x) + \hat{B}_h^*(x)^T]. \tag{4.11}$$

Assuming that $\nabla f(x)$ is already known, the forward version costs n gradient evaluations, while the central version costs the double. When an analytical expression of the gradient exists, its evaluation cost is typically a small multiple of that of the objective function (see Griewank, 1989). Formulae (4.9) and (4.10) are then cheaper to apply than (4.7). The recommended difference step lengths (4.5) and (4.6) apply also for the approximations (4.9) and (4.10), respectively.

We refer the reader to Sections 8.6 of Gill *et al.* (1981), Section 5.6 of Dennis and Schnabel (1983) and Subsection 8.4.3 of Conn *et al.* (2000) for a broader discussion on finite-difference methods.

4.2 Secant methods

Given an iterative procedure, secant methods update the approximated Hessian matrix from the previous one, using the curvature information collected between the two iterations. More precisely, the current Hessian approximation B_k is updated to B_{k+1} to enforce the secant equation

$$B_{k+1}s_k = y_k \tag{4.12}$$

where $s_k = x_{k+1} - x_k$ is the step between two successive iterates, and $y_k = g_{k+1} - g_k$ is the variation of the gradient along this step. This condition arises

from the mean-value theorem for vector-valued functions, which implies that the secant equation (4.12) is satisfied by the mean Hessian on the segment $[x_k, x_{k+1}]$, that is the matrix

$$\bar{B}_k \stackrel{\text{def}}{=} \int_0^1 \nabla^2 f(x_k + \alpha s_k)\, d\alpha. \tag{4.13}$$

Another interpretation is that equation (4.12) enforces the quasi-Newton model built (at iteration $k+1$) with such a matrix B_{k+1} to correctly interpolate the data f_{k+1}, g_k and g_{k+1}. The pair (s_k, y_k) is said to be the *secant pair* associated with equation (4.12). To avoid the cost of solving the quasi-Newton equation (2.15), the inverse matrix $H_k \stackrel{\text{def}}{=} B_k^{-1}$ is often recurred instead of B_k, using then the alternate secant equation for inverse Hessians:

$$H_{k+1} y_k = s_k. \tag{4.14}$$

The secant equation only imposes n linear constraints on the $n(n+1)/2$ degrees of freedom of the Hessian approximation (since it is a symmetric matrix). Many different secant approximation are therefore possible, depending on potential additional constraints that we would enforce. In Chapter 2, we have stressed on the important property of positive definiteness of the Hessian approximations to obtain descent directions for methods of linesearch type. We have also presented trust-region methods, which can yet deal with non positive definite Hessian approximation. We therefore distinguish in this introduction of secant updates the ones that do not enforce positive definiteness in Subsection 4.2.1, and the ones that do so in Subsection 4.2.2.

4.2.1 Indefinite secant updates

Broyden's update. — Broyden (1965) first used the secant equation to update Jacobian matrices (which may not be symmetric). He considers that the approximation B_k should not change too much between two consecutive iterations, and therefore define the updated approximation B_{k+1} as the matrix that minimizes

$$\|B_{k+1} - B_k\|_F \tag{4.15}$$

under the secant equation constraint, yielding the update formula

$$B_{k+1} = B_k + \frac{r_k s_k^T}{\|s_k\|^2}, \tag{4.16}$$

where $r_k \stackrel{\text{def}}{=} y_k - B_k s_k$ is the residual on the secant equation at B_k. This variational approach is at the basis of mostly every secant updates, each one depending on the norm used to measure the distance between B_k and B_{k+1}, and on the additional constraints imposed on B_{k+1}.

Powell-symmetric-Broyden update. — As the approximated Hessian matrix must be symmetric, Powell (1970) proposed a symmetric version of the Broyden update, known as the Powell-symmetric-Broyden (PSB) update, given by

$$B_{k+1} = B_k + \frac{r_k s_k^T + s_k r_k^T}{\|s_k\|^2} - \frac{\langle r_k, s_k \rangle}{\|s_k\|^4} s_k s_k^T, \tag{4.17}$$

where B_0 should be chosen as a symmetric matrix to enforce the symmetry of each approximation B_{k+1}. This formula gives in fact the limit of the matrix sequence starting from B_k, whose even elements are obtained by projecting the previous one on the space of symmetric matrices, and whose odd elements are obtained by projecting the previous one on the space of matrices fulfilling (4.12). Dennis and Moré (1977) show that this update is also simply obtained by minimizing the Frobenius norm of the correction $E_k \overset{\text{def}}{=} B_{k+1} - B_k$ under the secant equation and symmetry constraints.

Symmetric rank-one update. — Nowadays, the most widely used indefinite Hessian update is the symmetric-rank-one (SR1) formula, suggested independently by Broyden (1967), Davidon (1968), Fiacco and McCormick (1968), Murtagh and Sargent (1969), and Wolfe (1968). It is defined by

$$B_{k+1} = B_k + \frac{r_k r_k^T}{\langle r_k, s_k \rangle} \qquad \text{and} \qquad H_{k+1} = H_k + \frac{p_k p_k^T}{\langle p_k, y_k \rangle}, \tag{4.18}$$

where $p_k \overset{\text{def}}{=} s_k - H_k y_k$ is the residual on the inverse secant equation at H_k, and where B_0 (H_0) should also be chosen as a symmetric matrix to enforce the symmetry of each approximation B_{k+1} (H_{k+1}). This is the sole symmetric update that consists in a rank-one correction, hence its name. When the denominator of the correction in (4.18) is small, in the sense that

$$|\langle r_k, s_k \rangle| < \kappa_{\text{sr1}} \|r_k\| \|s_k\|, \tag{4.19}$$

for some small constant $\kappa_{\text{sr1}} \in (0, 1)$ (for instance $\kappa_{\text{sr1}} = 10^{-8}$), the update should not be performed (see Section 6.2 of Nocedal and Wright, 2006). Indeed, the curvature information along s_k provided by B_k is then already that of $\bar{B}_k$, meaning that no correction is needed (see also the analysis on the SR1 update behaviour by Conn, Gould and Toint, 1991)

The SR1 update is particularly suited for trust-region methods, which can exploit directions of negative curvature (see in particular Conn, Gould and Toint, 1988*b*). In this case, it is important to update the Hessian even at unsuccessful iterations, since such failures may result from a poor curvature information contained by the current Hessian approximation.

4.2.2 Positive definite secant updates

Davidon-Fletcher-Powell update. — Following the proposal made by Davidon (1959), Fletcher and Powell (1963) further studied and implemented the DFP update

$$B_{k+1} = (I - \rho_k y_k s_k^T) B_k (I - \rho_k s_k y_k^T) + \rho_k y_k y_k^T, \tag{4.20}$$

where $\rho_k \stackrel{\text{def}}{=} \langle y_k, s_k \rangle^{-1}$, or equivalently

$$H_{k+1} = H_k - \frac{H_k y_k y_k^T H_k}{\langle y_k, H_k y_k \rangle} + \rho_k s_k s_k^T. \tag{4.21}$$

It readily follows from this formula that B_{k+1} (H_{k+1}) remains positive definite if B_k (H_k) is positive definite and if the secant pair (s_k, y_k) verifies

$$\langle s_k, y_k \rangle > 0, \tag{4.22}$$

a condition that one can always enforce in the linesearch procedure if the objective function is bounded below (see Dennis and Schnabel, 1983), in particular by the curvature condition (2.23b).

Broyden-Fletcher-Goldfarb-Shanno update. — Developed concurrently by Broyden (1970), Fletcher (1970), Goldfarb (1970), and Shanno (1970), the BFGS update is presently considered as the most effective of all secant updates (at least for optimization methods unable to exploit directions of negative curvature). Its expression is given by

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{\langle s_k, B_k s_k \rangle} + \rho_k y_k y_k^T \tag{4.23}$$

or, using the Sherman-Morrison-Woodbury formula, by

$$H_{k+1} = (I - \rho_k s_k y_k^T) H_k (I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T. \tag{4.24}$$

As the DFP update, the BFGS update maintains symmetry positive definiteness of the sequence of approximated (inverse) Hessian provided that condition (4.22) holds at every iteration. Note also that the BFGS and DFP formulae are the "dual" of each other, in the sense that the changes $B \leftrightarrow H$ and $s \leftrightarrow y$ map one on the other. Note also that the first term of the correction in (4.23) removes any curvature in direction s_k, while the second term adjusts it to correspond to the gradient variation y_k. We still mention that a procedure for updating the Cholesky factor of B_k can be found in Goldfarb (1976).

4.3 Automatic differentiation

Automatic differentiation, which is sometimes alternatively called *algorithmic differentiation*, is a class of methods for numerically computing the derivatives of a function implemented in a computer program. The practical evaluation of such a function can be decomposed in a sequence of elementary operations, each one being trivial to differentiate. These elementary derivatives, evaluated at a particular point, are combined in accordance with the chain rule (1.37) to obtain the (exact) derivatives of the function. Contrary to symbolic differentiation (that is the way humans commonly compute by hand derivatives), no global expression of the derivatives is first determined and then evaluated at the point of interest. Applying automatic differentiation to a function thus results in a program, but not in a formula.

Since we focus in the following chapters on methods from the two previous classes, we elaborate no more on the subject and instead refer the interested reader to Griewank and Walther (2008) for a comprehensive coverage.

Chapter 5

Sparse Hessian approximation

In this chapter, we assume that the objective function $f : \mathbb{R}^n \to \mathbb{R}$ is sparse, in the sense that its Hessian matrix is sparse, which means that it has a few number of potential nonzero entries. The indices of these entries constitute the *sparsity pattern* $\mathcal{S}$ of the function. Such functions arise for instance from the discretization of infinite-dimensional problems on a grid (remember the examples in Subsection 3.1.1). We also focus on large-scale problems, that is those for which the problem size n is large, because the exploitation of sparsity is then more critical.

When building a Hessian approximation of such a function f, it would be interesting to preserve the Hessian sparsity pattern, and we obviously do not wish to take time computing its known zero entries. It is the purpose of this chapter to present some existing and new methods to achieve this goal, and then to compare them numerically inside the framework of the RMTR method (see Section 3.2).

Finite-difference methods from Subsection 4.1.2 normally preserve the sparsity pattern of the Hessian, but require n gradient evaluations (which makes this approach practically unaffordable when n is large) as they compute all the zero entries (up to the method precision). Powell and Toint (1979), inspired by Curtis, Powell and Reid (1974), propose to use the sparsity pattern to drastically reduce this number of evaluations. Their approach is presented in Section 5.1, together with an improvement by Goldfarb and Toint (1984), but we also refer to Coleman and Moré (1984), Coleman, Garbow and Moré (1985), Hossain and Steihaug (1998, 2002), Gebremedhin, Manne and Pothen (2005) for further developments along this line of research.

In the secant methods presented in Section 4.2, the approximated Hessian is updated by some low-rank correction, destroying the Hessian sparsity pattern, which is not acceptable for the considered class of problems. This deficiency may be addressed in several ways. The first is to construct a Hessian ap-

proximation subject to the secant equation (4.12) but preserving the sparsity pattern. This can be achieved by applying the sparse PSB update derived by Marwil (1978) and Toint (1977), in which the new approximation is chosen to minimize the size of the correction (see Section 5.2 below). A second possibility is to consider the use of *partitioned updating* for partially separable functions, as proposed in Griewank and Toint (1982*b*,*c*) (see also Griewank and Toint (1982*a*) and Griewank (1991) for the convergence theory, as well as Toint (1983) and Griewank and Toint (1984*a*) for numerical experiments). The interested reader may also consult the works of Plitt (1981) and Bock and Plitt (1984). In this technique, the classical updating process is not applied on the Hessian of the objective function f, but on that of each of its element functions f_i. However this requires the knowledge of the element functions' gradients $\nabla f_i(x)$, which is not always realistic. Consider, for instance, a discretized problem arising from partial differential equations whose gradient is computed by solving an adjoint equation. In this case, the system solution provides the full gradient vector, but does not allow the distinction between gradients of the involved element functions, which makes the direct exploitation of the problem's partially separable structure impossible. We hence show in Section 5.3 how the partially separable nature of the objective function can nevertheless be used, even if only the full gradient is available. Third, note that neither of the two previous updating schemes enforce positive definite Hessian approximations. This may be considered as a drawback, especially in the context of multilevel methods, whose efficiency is best on convex problems. We therefore complete our panel of Hessian approximation methods with a new attempt to obtain a positive definite partitioned updating method (Section 5.4), whose performance is then included in our comparison.

Finally, note that an important component of the RMTR algorithm, both in theory and numerical performance, is that it uses smoothing strategies in which individual Hessian entries must be available for this procedure to be well-defined. As a consequence, an approach that would be based on computing Hessian-times-vector products would, in our context, be relatively inefficient. Moreover, techniques using Hessian-times-vector products are notoriously difficult to precondition. We therefore do not consider methods (mentioned in Section 4.3) nor truncated Newton approaches (see for instance Nash, 2000*b*) because they typically rely on such techniques.

To avoid clumsy notations, we do not mention the outer iteration indices in this chapter. For instance, the iterate $x_{i,k}$ in the RMTR algorithm is simply denoted by x. Hence, we consider in Section 5.1 the design of an approximation B of the Hessian of the function f at some point x. In the rest of the chapter, we consider instead the update of the Hessian approximation B at the iterate x to an approximation B^+ at the next iterate x^+, the "$^+$" superscript being used to denote objects at the next iteration.

5.1 Sparse finite-difference methods

5.1.1 Lower-triangular substitution method

Finite-differences can be used to approximate derivatives of the objective function. Yet it would be inefficient to compute many zero values in a Hessian known to be sparse. This observation is at the basis of the CPR algorithm developed for Jacobian matrices by Curtis *et al.* (1974). In the absence of sparsity, each column of the Jacobian is computed by taking a finite difference in the corresponding direction of the canonical basis of $\mathbb{R}^n$. Instead, their idea is to combine as many as possible of these directions into one, such that the differences can be untangled (the sparsity patterns of the assembled columns do not overlap). This procedure has then been refined for (symmetric) Hessian matrices by Powell and Toint (1979). In the most economical variant (the substitution method), only the lower (or upper) triangular part of the matrix is computed. This method does not directly provide the values of the Hessian elements, but these are given as the solution of a triangular system of linear equations. The procedure computes first the column grouping by applying the CPR algorithm on the (permuted) lower triangular part of the Hessian as stated in Algorithm 5.1.

Algorithm 5.1 (Lower-triangular substitution method: LTS)

Step 1. Permute the rows and columns of B to minimize the maximum number of nonzero entries in any row of B.

Step 2. Apply the CPR algorithm to the lower triangular sparsity pattern of the permuted B.

Step 3. Compute the corresponding gradient differences.

Step 4. Reconstruct the entries of the estimated B by solving a triangular system of equations.

We now describe these steps more formally, considering the evaluation of the sparse Hessian B of a function $f : \mathbb{R}^n \to \mathbb{R}$ at some point x, and give some details on our numerical implementation of the LTS method (that constitutes the Fortran LTS package; see Chapter 8 for further details).

As also recommended in the case of sparse Jacobian (see Curtis *et al.*, 1974), the rows and columns of B are first ordered in Step 1 to minimize the maximum number of nonzero entries (according to the sparsity pattern) in any row of B. We follow the procedure proposed by Powell and Toint (1979): letting j decrease from n to 2, we iteratively look for a row i of the leading $j \times j$ submatrix of H that contains the fewest nonzero entries and then

exchange the i-th and j-th rows and columns of B. No permutation of rows or columns is actually performed in the LTS package, but we instead adjust an array w of pointers that indicates the row and column ordering that would be obtained. When entering the j-th iteration, the first j entries of w contain the row numbers of the current leading $j \times j$ submatrix of B sorted by decreasing number of nonzero entries, while $w(k)$ (for $j < k \leqslant n$) holds the actual row (and column) number of the k-th row (and column) of the permuted Hessian.

Using the CPR algorithm developed by Curtis *et al.* (1974), the columns of the triangular lowerpart of the (permuted) Hessian are then iteratively gathered in p groups in Step 2 such that no two columns from the same group have nonzero entries on the same row. In this algorithm, the set $\mathcal{C}_k$ gathers the indices of the columns belonging to the first k groups. At iteration k, we sort the columns not belonging to $\mathcal{C}_{k-1}$ in decreasing order of their number of unknown entries (that is nonzero entries whose row number is not in $\mathcal{C}_{k-1}$). To form the k-th group, we consider successively the columns of this sequence, and add the currently considered column to the k-th group if it has no unknown entry on the same rows as columns already belonging to this group.

We then compute in Step 3 the gradient differences corresponding to these p groups:

$$y^\ell = \nabla f(x + h^\ell) - \nabla f(x) \qquad \text{for } \ell = 1, \ldots, p$$

where the component h_j^ℓ is a nonzero difference step length if column j belongs to group ℓ, and zero otherwise. In the LTS package, we follow the rule (4.5) to define these nonzero difference step lengths h_j^ℓ.

We finally consider in Step 4 the linear system of equations

$$Bh^\ell = y^\ell \quad \text{for} \quad \ell = 1, \ldots, p$$

in the variables B_{ij} with $(i, j) \in \mathcal{S}$ and $i \geqslant j$, since B is symmetric. If these variables are sorted in the lexicographical order, this system may be solved with a backward substitution method (see Powell and Toint, 1979). We therefore consider successively the rows of the (permuted) Hessian starting from the last one. At each row i, we compute its unknown entries (and consequently that of column i), and then implicitly update the linear system by adjusting the corresponding gradient differences:

$$B_{ij} = y_i^{\ell(j)} / h_j^{\ell(j)} \quad \text{and} \quad y_j^{\ell(i)} \leftarrow y_j^{\ell(i)} - B_{ij} h_i^{\ell(i)}$$

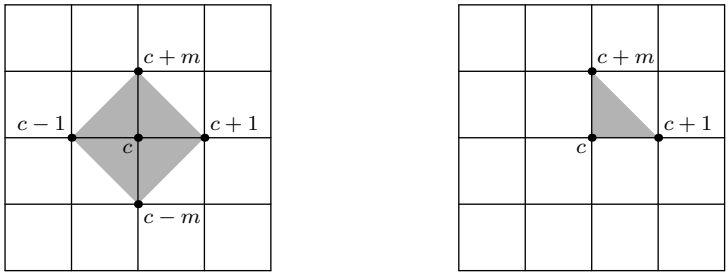
for each j such that $(i, j) \in \mathcal{S}$ and $i \geqslant j$, and where $\ell(j)$ is the group that contains column j. This completes the description of the LTS method.

Still note that the first two steps should only be performed once to define the column groups, since they depend on the sparsity pattern $\mathcal{S}$ of the Hessian, but not on the point x at which it is evaluated.

5.1.2 Optimal column grouping and covering molecules

Even though a counterexample can be found, the column grouping resulting from Steps 1 and 2 of Algorithm 5.1 is often nearly optimal in terms of the necessary number of gradient evaluations per Hessian approximation. On the other hand, Goldfarb and Toint (1984) describe optimal groupings based on computational molecules or stencils typically arising from the discretization of differential equations.

When partial differential equations are discretized using a finite-difference approximation, sparsity pattern $\mathcal{S}$ of the resulting problem is completely determined by the *computational molecule* or *stencil* of the finite-difference operator, the variables ordering (*i.e.* the numbering of the mesh points), and the given boundary conditions. The stencil of an operator indicates which (neighbour) variables are tangled with a given one. For instance, consider the problem (3.5), which uses the two-dimensional five-point Laplacian operator on a rectangular region with a $m \times q$ mesh numbered in the standard way. The stencil centred at the a given point c then covers in addition the four points directly on the North $(c+m)$, South $(c-m)$, West $(c-1)$ and East $(c+1)$ of the considered point, as illustrated by Figure 5.1a. The only potential nonzero entries in the c-th row (column) of the Hessian matrix then occur in columns (rows) $c-m$, $c-1$, c, $c+1$ and $c+m$.



(a) *Molecule.* (b) *Forward molecule.*

Figure 5.1 — *Molecules for the 5-point Laplacian operator.*

In the LTS method, only the pattern for the lower-triangular part of the Hessian is relevant. It corresponds to the part of the molecule that covers mesh points with indices larger than that of its centre. Goldfarb and Toint (1984) refer to these partial molecules as *forward computational molecules*. If we consider again the former example, its forward molecule is now a triangle covering mesh points c, $c+1$ and $c+m$, as shown in Figure 5.1b.

Partitioning the columns of the lower-triangular part of Hessian into a minimal number of suitable groups now becomes a simple task, given the forward molecule. First, we completely cover all mesh points by disjoint molecules. Each group of columns then gathers all columns whose indices are those of

variables occupying the same position in every molecule of the cover. This is clearly a valid column grouping, since all the molecules used to form the cover are disjoint. In our example, a suitable covering of the mesh is shown in Figure 5.2. This cover yields three column groups defined by

$$\ell(i + (j-1)m) \equiv i + 2j \pmod 3,$$

for $i = 1, \ldots, m$ and $j = 1, \ldots, q$.

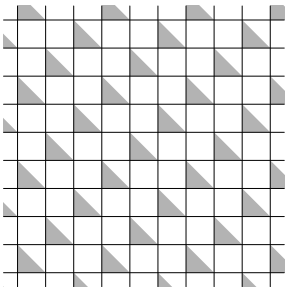


Figure 5.2 — *Forward molecule and cover for a 5-point Laplacian operator.*

5.2 Sparse secant updating

In the domain of secant Hessian approximations, Marwil (1978) and Toint (1977) developed a sparse Hessian updating process for which a global and superlinear local convergence theory is established when combined with trust-region techniques (see Toint, 1979). In this method, the updated Hessian B^+ is required to be symmetric and to fulfil the sparsity pattern:

$$B^+ = {B^+}^T \text{ and } \mathcal{P}(B^+) = B^+, \tag{5.1}$$

where $\mathcal{P}$ is the *gangster* operator (see for instance Powell and Toint, 1981) that zeroes all entries of a matrix according to the sparsity pattern $\mathcal{S}$, that is

$$[\mathcal{P}(A)]_{ij} = \begin{cases} A_{ij} & \text{if } (i,j) \in \mathcal{S}, \\ 0 & \text{otherwise.} \end{cases} \tag{5.2}$$

for every matrix $A \in \mathbb{R}^{n \times n}$. We also require the secant equation (4.12) to be verified. As these conditions do not fully determine the updating process, a standard technique (see Dennis and Schnabel, 1983) consists in choosing the smallest correction matrix (in an appropriate norm) such that the updated

Hessian satisfies (5.1) and (4.12). We consider here the weighted Frobenius norm defined by

$$\|A\|_\Omega^2 \stackrel{\text{def}}{=} \sum_{j,l=1}^{n} \Omega_{jl} A_{jl}^2,$$

for some $n \times n$ weight matrix Ω with positive elements. These considerations lead to the *Sparse Powell-symmetric-Broyden* update presented in Algorithm 5.2, where $\bullet$ denotes the Hadamard product, and $\cdot^{\Pi}$ the inverse for that operation (see Subsection 1.1.2).

Algorithm 5.2 (Sparse PSB update: S-PSB)

The current Hessian approximation B, the step s, the corresponding gradient variation y and a weight matrix Ω are given.

Step 1. Define $S = \mathcal{P}(ss^T) \bullet \Omega^{\Pi} + \text{diag}(\mathcal{P}(\Omega^{\Pi})(s \bullet s))$.

Step 2. Solve $S\lambda = y - Bs$ for λ.

Step 3. Compute $B^+ = B + \mathcal{P}(s\lambda^T + \lambda s^T) \bullet \Omega^{\Pi}$.

The linear system in Step 2 may be quite efficiently solved by a preconditioned conjugate gradient method (see Algorithm 2.8). The chosen preconditioner is given by the diagonal term $\text{diag}(\mathcal{P}(\Omega^{\Pi})(s \bullet s))$.

5.3 Partially separable secant updating

Assume now that the objective function is partially separable. As a consequence of (1.33), we immediately deduce that

$$\nabla f(x) = \sum_{i=1}^{m} \nabla f_i(x) \quad \text{and} \quad \nabla^2 f(x) = \sum_{i=1}^{m} \nabla^2 f_i(x), \tag{5.3}$$

where the vectors $\nabla f_i(x)$ and matrices $\nabla^2 f_i(x)$ have only a small number of potentially nonzero components: those corresponding to the variables indexed by $\mathcal{I}_i$. The set $\mathcal{I}_i^c$ is the complementary of $\mathcal{I}_i$, that is $\{1, \ldots, n\} \setminus \mathcal{I}_i$. We denote by $I_i \in \mathbb{R}^{n \times n}$ the identity matrix with zeros at diagonal elements not indexed by $\mathcal{I}_i$, and define $J_i \in \mathbb{R}^{n \times n}$ to be the $n \times n$ matrix with zero entries everywhere except in positions $\mathcal{I}_i \times \mathcal{I}_i$ where they are equal to 1.

Partitioned updating. — The partitioned updating technique by Griewank and Toint (1982*b*) then updates every element Hessian H_i separately, using the

element secant equation

$$B_i^+ s_i = y_i \stackrel{\text{def}}{=} g_i^+ - g_i \tag{5.4}$$

where B_i^+ is the updated approximation of $\nabla^2 f_i(x^+)$, $s_i \in \mathbb{R}^n$ is the step vector s whose components not in $\mathcal{I}_i$ have been zeroed, $g_i = \nabla f_i(x)$ and $g_i^+ = \nabla f_i(x^+)$. However, as we indicated in the introduction, assuming the knowledge of the element gradients g_i and g_i^+ and their difference y_i may be unrealistic, and we are therefore interested in a technique that would avoid this requirement.

Design of the new update. — As in the previous section, we use variational properties to elaborate our new update and each matrix B_i^+ is chosen as close as possible to the current approximation B_i, under suitable conditions. More precisely, we propose to compute a correction to $B = \sum_{i=1}^m B_i$ and a set of element gradient differences y_i which together minimize some (possibly) weighted sum (with weights $\omega_i > 0$) of the squared distances between B_i^+ and B_i (in the Frobenius norm)

$$\tfrac{1}{2} \sum_{i=1}^m \omega_i \|B_i^+ - B_i\|_F^2, \tag{5.5}$$

under the structure constraints on the element gradients and Hessians

$$I_i y_i = y_i \quad \text{and} \quad J_i \bullet B_i^+ = B_i^+ \qquad (i = 1, \ldots, m), \tag{5.6}$$

the symmetry of the Hessian corrections $E_i \stackrel{\text{def}}{=} B_i^+ - B_i$, that is

$$E_i = E_i^T \qquad (i = 1, \ldots, m) \tag{5.7}$$

and the constraint that the (unknown) element gradients differences y_i sum up to the full gradient difference y, that is

$$\sum_{i=1}^m y_i = y. \tag{5.8}$$

Obviously, we also require that the element secant equation (5.4) holds for all $i = 1, \ldots, m$. We stress that no assumption is made here on the positive semi-definiteness of the element Hessians B_i.

The solution of this variational problem (5.4)–(5.8) is obtained by a Lagrangian technique. The Lagrangian function of this problem, which depends on multipliers $\nu_i \in \mathbb{R}^n$ and $\chi_i \in \mathbb{R}^{n \times n}$ $(i = 1, \ldots m)$ for constraints (5.6), $\mu \in \mathbb{R}^n$ for the constraint (5.8) and $2\lambda_i \in \mathbb{R}^n$ $(i = 1, \ldots m)$ for constraints

(5.4), can be written as

$$\begin{aligned} L(E_1,\ldots,E_m,y_1,\ldots,y_m,\nu_1,\ldots,\nu_m,\chi_1,\ldots,\chi_m,\mu,\lambda_1,\ldots,\lambda_m) \\ = \sum_{i=1}^{m}\Big[\tfrac{1}{2}\omega_i\|E_i\|_F^2 - \langle\nu_i,(I_i-I)y_i\rangle - \langle\chi_i,(J_i-J)\bullet(B_i+E_i)\rangle - \langle\mu,y_i\rangle \\ - \Big\langle 2\lambda_i,\Big(B_i+\tfrac{E_i+E_i^T}{2}\Big)s_i - y_i\Big\rangle\Big] + \langle\mu,y\rangle \end{aligned} \tag{5.9}$$

where $\langle A,B\rangle = \mathrm{tr}(A^TB)$ is the usual inner product for matrices $A,B\in\mathbb{R}^{n\times n}$. Note that there is no multiplier for the symmetry constraints (5.7) on the correction E_i, because the contribution of secant equations to the Lagrangian is expressed in such a way that these constraints are automatically fulfilled.

At optimal solutions, the derivative of the Lagrangian with respect to each variable y_i must be zero, that is

$$-(I_i-I)\nu_i - \mu + 2\lambda_i = 0, \quad (i=1,\ldots,m). \tag{5.10}$$

Focusing now on the components indexed by $\mathcal{I}_i$, we observe that the first term of (5.10) disappears, yielding $2\tilde{\lambda}_i = \tilde{\mu}_i$, where $\tilde{\mu}_i,\tilde{\lambda}_i\in\mathbb{R}^{n_i}$ is the part of μ and λ_i, respectively, corresponding to the components indexed by $\mathcal{I}_i$. Clearly, these equations imply that every $\tilde{\lambda}_i$ is in fact a (maybe overlapping) piece of vector $\lambda \stackrel{\text{def}}{=} \mu/2$.

The derivatives of the Lagrangian with respect to variables E_i also have to be zero, which means that the directional derivative of L with respect to E_i is zero in every (matrix) direction $K_i\in\mathbb{R}^{n\times n}$:

$$\begin{aligned} \mathrm{D}_{E_i}L\cdot K_i &= \tfrac{1}{2}\omega_i\mathrm{D}_{E_i}\,\mathrm{tr}(E_i^TE_i)\cdot K_i - \langle\chi_i,(J_i-J)\bullet K_i\rangle - \lambda_i(K_i+K_i^T)s_i \\ &= \omega_i\,\mathrm{tr}\,K_i^TE_i - \langle\chi_i,(J_i-J)\bullet K_i\rangle - \lambda_i(K_i+K_i^T)s_i = 0, \end{aligned} \tag{5.11}$$

where $\mathrm{D}_{E_i}L\cdot K_i$ is the derivative of L with respect to E_i in direction K_i. In particular, we may choose K_i as any matrix with the same structure as B_i, for which the only potentially nonzero entries are in the submatrix indexed by $\mathcal{I}_i\times\mathcal{I}_i$. We denote this $n_i\times n_i$ submatrix by $\tilde{K}_i$. For this choice of K_i, the second term of equation (5.11) disappears when considering only entries indexed by $\mathcal{I}_i\times\mathcal{I}_i$, and we obtain that

$$\omega_i\,\mathrm{tr}(\tilde{K}_i^T\tilde{E}_i) = \tilde{\lambda}_i(\tilde{K}_i+\tilde{K}_i^T)\tilde{s}_i,$$

where $\tilde{E}_i\in\mathbb{R}^{n_i\times n_i}$ is formed with the elements of E_i indexed by $\mathcal{I}_i\times\mathcal{I}_i$. Developing this expression, we find that

$$\sum_{k,l=1}^{n_i}[\tilde{K}_i]_{jl}\left([\tilde{E}_i]_{jl} - \omega_i^{-1}\left([\tilde{\lambda}_i]_j[\tilde{s}_i]_l + [\tilde{\lambda}_i]_l[\tilde{s}_i]_j\right)\right) = 0, \quad \forall\,\tilde{K}_i\in\mathbb{R}^{n_i\times n_i},$$

which implies that

$$\tilde{E}_i = \omega_i^{-1} \left(\tilde{\lambda}_i \tilde{s}_i^T + \tilde{s}_i \tilde{\lambda}_i^T \right). \tag{5.12}$$

On the other hand, the element Hessian matrices B_i^+ have to fulfil the secant equation (5.4), *i.e.* $y_i = B_i s_i + E_i s_i$, yielding $p_i \stackrel{\text{def}}{=} y_i - B_i s_i = E_i s_i$. Focusing again on the components indexed by $\mathcal{I}_i$, and using (5.12), we have that

$$\tilde{p}_i = \tilde{E}_i \tilde{s}_i = \omega_i^{-1} \left(\tilde{\lambda}_i \tilde{s}_i^T \tilde{s}_i + \tilde{s}_i \tilde{\lambda}_i^T \tilde{s}_i \right) = \omega_i^{-1} \big(\|\tilde{s}_i\|^2 \tilde{I}_i + \tilde{s}_i \tilde{s}_i^T \big) \tilde{\lambda}_i \stackrel{\text{def}}{=} \omega_i^{-1} \tilde{S}_i \tilde{\lambda}_i,$$

where $\tilde{p}_i \in \mathbb{R}^{n_i}$ is the subvector of p_i indexed by $\mathcal{I}_i$. From the small matrix $\tilde{S}_i \in \mathbb{R}^{n_i \times n_i}$, we construct the matrix $S_i \in \mathbb{R}^{n \times n}$ by adding zero rows and columns in such a way that S_i has the same structure as B_i. Consequently, $p_i = \omega_i^{-1} S_i \lambda$ and, using the decomposition of the gradient difference y into element gradient differences y_i, we obtain that

$$p \stackrel{\text{def}}{=} \sum_{i=1}^{m} p_i = y - Bs = \left(\sum_{i=1}^{m} \omega_i^{-1} S_i \right) \lambda \stackrel{\text{def}}{=} S\lambda.$$

Therefore, we just need to solve the system $p = S\lambda$, then substitute its solution λ in (5.12) to find the corrections E_i and finally, determine the updated element Hessians $B_i^+ = B_i + E_i$ and the full Hessian $B^+ = B + \sum_{i=1}^m E_i$.

Note that this updating procedure requires explicitly neither the element gradients g_i^+ or g_i, nor their differences y_i, despite that the latter appear as variables in our variational problem. There is thus no reason to compute them at each iteration, leading to Algorithm 5.3.

Algorithm 5.3 (Partially separable PSB updating: PS-PSB)

The current Hessian approximation B, the step s and the corresponding gradient variation y, and a set of weights ω_i are given.

Step 1. Decompose the vector s into $\{s_i\}_{i=1}^m$ and define the matrix $S = \sum_{i=1}^m \omega_i^{-1} (\|s_i\|^2 I_i + s_i s_i^T)$.

Step 2. Solve the system $S\lambda = y - Bs$ for λ and decompose it into $\{\lambda_i\}_{i=1}^m$.

Step 3. Update the Hessian: $B^+ = B + \sum_{i=1}^m \omega_i^{-1} (\lambda_i s_i^T + s_i \lambda_i^T)$.

As in the previous algorithm, the linear system in Step 2 may be quite efficiently solved by a preconditioned conjugate gradient method (see Algorithm 2.8) with a zero starting point. The chosen preconditioner is, this time, given by the diagonal term $\sum_{i=1}^m \omega_i^{-1} \|s_i\|^2 I_i$.

Link between sparse and partially separable PSB updates. — The new Algorithm 5.3 appears to be very similar to Algorithm 5.2. In fact, they turn out to differ only by the choice of the norm used in the variational problem, the partially separable version giving more weight to the parts of the Hessian belonging to overlapping elements. It may nevertheless be interpreted as a "PSB-type" update where one minimizes a (weighted) Frobenius norm of the matrix updates subject to linear conditions.

To be more precise, both algorithms produce the same update if we defined the weight matrix Ω in Algorithm 5.2 such that

$$\Omega^{\maltese} = \sum_{i=1}^{m} \omega_i^{-1} J_i. \tag{5.13}$$

Indeed, for the particular choice of Ω, the matrix S is identical for both considered algorithms:

$$\begin{aligned} S &= \mathcal{P}\left(ss^T \bullet \sum_{i=1}^{m} \omega_i^{-1} J_i\right) + \operatorname{diag}\left(\mathcal{P}\left(\sum_{i=1}^{m} \omega_i^{-1} J_i\right)(s \bullet s)\right) \\ &= \mathcal{P}\left(\sum_{i=1}^{m} \omega_i^{-1}(ss^T \bullet J_i)\right) + \sum_{i=1}^{m} \omega_i^{-1} \operatorname{diag}\left(J_i(s \bullet s)\right) \\ &= \sum_{i=1}^{m} \omega_i^{-1} s_i s_i^T + \sum_{i=1}^{m} \omega_i^{-1} \operatorname{diag}\left(\|s_i\|^2 b_i\right) = \sum_{i=1}^{m} \omega_i^{-1}\left(s_i s_i^T + \|s_i\|^2 I_i\right), \end{aligned}$$

where $b_i = \operatorname{diag}(I_i)$ is a vector with value 1 for components indexed by $\mathcal{I}_i$ and value 0 elsewhere. It now remains to show that the correction $E \stackrel{\text{def}}{=} B^+ - B$ is the same in both cases:

$$\begin{aligned} E &= \mathcal{P}\left((s\lambda^T + \lambda s^T) \bullet \sum_{i=1}^{m} \omega_i^{-1} J_i\right) = \mathcal{P}\left(\sum_{i=1}^{m} \omega_i^{-1}(s\lambda^T + \lambda s^T) \bullet J_i\right) \\ &= \sum_{i=1}^{m} \omega_i^{-1}\left(s_i \lambda_i^T + \lambda_i s_i^T\right). \end{aligned}$$

Convergence properties. — As a consequence of this result, the convergence theory presented by Toint (1979) (in the unweighted case) immediately adapts to provide global and local convergence with superlinear speed for Algorithm 5.3 embedded inside a trust-region method.

5.4 Positive definite partially separable secant updating

In the previous sections, positive definiteness of the updated Hessian was not imposed. But as we aim to design Hessian updating procedures in the frame-

work of multilevel methods, this property could be advantageous since multigrid methods are known to perform well on convex problems. However, Sorensen (1981) exhibited a counterexample which shows that it may be impossible to require at the same time the preservation of a sparsity pattern for the updated Hessian, its positive definiteness and the secant equation (4.12). On the other hand, Toint (1981*a*) gave sufficient conditions for the existence of such updates.

Despite these hardly auspicious considerations, we attempt to design such a positive definite Hessian updating process in the specific context of partially separable unconstrained optimization, but still without assuming the knowledge of the individual element gradients. We start from the observation that, if these individual gradients were available, then we could apply the partitioned updating of Griewank and Toint (1982*b*) and perform the BFGS update on each block of the Hessian matrix. Hence, the idea is to re-create a collection of "element gradients" whose sum equals the full gradient. These vectors are then used in the partitioned BFGS update. However, as each elemental secant equation automatically holds for each elemental BFGS correction, the global secant equation then also holds, which we know is impossible in general. Our proposal is then to relax the summation condition (5.8) whenever necessary, in which case our element gradients may differ from their analytical values and the global secant equation no longer holds exactly.

In addition, we should ensure that $\langle s_i, y_i \rangle$ is positive for the BFGS formula (4.23) to generate a positive definite update B_i^+. Splitting the gradient, and thus y, blindly could consequently be inappropriate. We thus aim to split the gradient variation y into elemental gradient variations y_i such that their sum is (not too different of) y and that some elemental "curvature" is maintained in the sense that

$$\xi_i = \xi_i(y_i) \stackrel{\text{def}}{=} \frac{\langle s_i, y_i \rangle}{\langle s_i, B_i s_i \rangle} \geqslant \frac{\epsilon}{m} \sum_{j=1}^{m} \frac{\langle s_j, y_j \rangle}{\langle s_j, B_j s_j \rangle} \stackrel{\text{def}}{=} \epsilon \bar{\xi} \tag{5.14}$$

for some fixed $\epsilon > 0$. Note that we assume that each B_i have been kept positive definite, so $\langle s_i, B_i s_i \rangle > 0$. Remark also that due to the structure of y_i and B_i, the inner products $\langle s_i, y_i \rangle$ and $\langle s_i, B_i s_i \rangle$ can equivalently be written as $\langle s, y_i \rangle$ and $\langle s, B_i s \rangle$, respectively

5.4.1 Uniform splitting

The simplest splitting procedure is to split the vector y uniformly, in the following sense. Consider the ℓ-th component of y and consider the collection $\{y_i\}_{i \in \mathcal{K}(\ell)}$ where $\mathcal{K}(\ell) = \{i \mid \ell \in \mathcal{I}_i\}$ (this collection gathers the particular y_i whose ℓ-th component is possibly nonzero). Then the ℓ-th component of each y_i in this collection is defined as the ℓ-th component of y divided by $|\mathcal{K}(\ell)|$. Although this procedure may be judged simplistic, we still add it to our algorithm test in the following form.

Algorithm 5.4 (Uniformly partitioned BFGS Update: UP-BFGS)

The current element Hessian approximation B_i, the step s, the corresponding gradient variation y, and a threshold $\kappa > 0$ are given.

Step 1. Decompose the vector s into $\{s_i\}_{i=1}^m$ and split uniformly y into $\{y_i\}_{i=1}^m$.

Step 2. Perform the BFGS update (4.23) on each B_i using the secant pair (s_i, y_i) provided that $\langle s_i, y_i \rangle \geqslant \kappa \langle s_i, B_i s i \rangle$.

5.4.2 Curvature flow problem

Unfortunately, the vectors y_i resulting from the uniform splitting do not necessarily satisfy the curvature constraints (5.14). We then attempt to modify this initial split of y by considering a *feasible flow problem*. We define a network whose nodes represent the elements of the partially separable structure. Each pair of nodes corresponding to elements sharing at least one variable is connected by an arc. Each of these arcs is directed to start at the node of the pair with largest value of the curvature measure ξ_i and terminate at the node of the pair with the smallest ξ_i. Each node i that fulfils (5.14) is then considered as a source of (curvature) flow in the network and, symmetrically, each node for which (5.14) fails is considered as a sink.

Our aim is then to modify the values of the y_i, which then results in corresponding modifications of the curvature values ξ_i, such that (5.14) holds for all nodes. This idea arises from Griewank and Toint (1984*b*), who studied the existence of a convex decomposition of partially separable functions (one in which each element function is convex). In that paper, they shift quadratic terms from some elements to others to build a decomposition at least locally convex. In our case, the modifications are achieved by performing *pushes* that shift curvature across the network from the sources to the sinks.

Performing a push. — The shift of curvature along the arc $i \to j$ is obtained by considering only the variables shared by elements i and j, that is those in the set $\mathcal{I}_i \cap \mathcal{I}_j$, which is nonempty by construction of the network. In what follows, we use the notation $\hat{x}$ to restrict vector x to the components indexed by this set $\mathcal{I}_i \cap \mathcal{I}_j$ (i and j remaining implicit), and $\check{x}$ to restrict it to the complementary set of variables, which is $\mathcal{I}_i^c \cup \mathcal{I}_j^c$. To improve ξ_j, we update the corresponding vector y_j by adding to its part $\hat{y}_j$ some subvector u such that $\langle \hat{s}, u \rangle > 0$. Hence, the updated y_j^+ may be decomposed in $\check{y}_j^+ = \check{y}_j$ and $\hat{y}_j^+ = \hat{y}_j + u$, which implies that

$$\left\langle s, y_j^+ \right\rangle = \left\langle \check{s}, \check{y}_j \right\rangle + \left\langle \hat{s}_j, \hat{y}_j + u \right\rangle > \left\langle \check{s}, \check{y}_j \right\rangle + \left\langle \hat{s}_j, \hat{y}_j \right\rangle = \langle s, y_j \rangle ,$$

and therefore that ξ_j^+ is larger than ξ_j after this modification. The same subvector u is then symmetrically subtracted from $\hat{y}_i$, with the effect that curvature is shifted from node i to node j. The amount of shifted curvature is limited by the amount of positive curvature available in node i. We also impose another bound whose purpose is to avoid a too large increase in the norm of the vectors y_i and y_j (which in turn results in an increase in the norm and condition number of the Hessian approximation), and require that the norms of $\hat{y}_i$ and $\hat{y}_j$ remain bounded above by some constant M. This constraint defines the capacity of the arc $i \to j$, that is the amount of curvature that can be shifted along this arc.

More precisely, suppose that we would like to transfer δ units of flow (in our case, curvature) along the arc $i \to j$, with the aim of satisfying demand at node j subject to capacity constraint on this arc. Consider the resulting modification of vectors $\hat{y}_i$ and $\hat{y}_j$ with respect to their mean $w = \frac{1}{2}(\hat{y}_i + \hat{y}_j)$. To preserve the summation condition (5.8), $\hat{y}_i^+ + \hat{y}_j^+$ should be equal to $\hat{y}_i + \hat{y}_j$, and so $w^+ = w$, yielding that

$$\hat{y}_i^+ = w - v \quad \text{and} \quad \hat{y}_j^+ = w + v, \tag{5.15}$$

with $v = \frac{1}{2}(\hat{y}_j - \hat{y}_i) + u$.

As indicated above, we require that the norms of these two vectors are bounded above by the constant M. First consider the case where $\hat{s}$ and w are linearly independent. Given that

$$\|w \pm v\|^2 \leqslant \|w\|^2 + \|v\|^2 + 2\,|\langle w, v\rangle|\,,$$

we choose v to zero the term $\langle w, v\rangle$ to limit the values of these norms. In this situation, consider a fixed improvement in ξ_j. The value of $\langle \hat{s}, v\rangle$ is then fixed, and the norm of $\hat{y}_j^+$ (which is now the same as that of $\hat{y}_i^+$) is minimal if the vector v lies in the plane Π spanned by $\hat{s}$ and w. Indeed, consider the orthogonal decomposition of v as

$$P_\Pi v + (I - P_\Pi)v,$$

where P_Π is the orthogonal projection on Π. Then, the squared norm

$$\|y_j^+\|^2 = \|w\|^2 + \|P_\Pi v\|^2 + \|(I - P_\Pi)v\|^2 \tag{5.16}$$

has to be minimal with $\langle \hat{s}, v\rangle$ fixed and $\langle w, v\rangle$ equal to zero (due to the orthogonality assumption on v and w). But, $\langle \hat{s}, v\rangle = \langle \hat{s}, P_\Pi v\rangle$ and $\langle w, v\rangle = \langle w, P_\Pi v\rangle$ because $\hat{s}$ and w lie in Π. So the two constraints determine only (but completely) the component of v lying in the plane Π. The minimum of (5.16) is thus reached when $(I - P_\Pi)v$ is zero. We now give an explicit expression of vector v, knowing that it lies in plane Π and is perpendicular to w. Observe that

these conditions define a one-dimensional space and are fulfilled by $(I - P_w)\hat{s}$. Therefore,

$$v = \beta \left(I - \frac{ww^T}{\langle w, w \rangle} \right) \hat{s} \tag{5.17}$$

for some scalar β.

Now consider the other case where $\hat{s}$ and w are collinear. Choosing v orthogonal to w (and thus to $\hat{s}$) prevents increasing ξ_j. Then, we just set

$$v = \beta \hat{s} \tag{5.18}$$

for some scalar β, enabling a direct increase of $\langle \hat{s}, v \rangle$, and thus of ξ_j.

In both cases, the positive scalar β is determined such that

$$\|\hat{y}_i^+\| \leqslant M \quad \text{and} \quad \|\hat{y}_j^+\| \leqslant M, \tag{5.19}$$

and that $\langle \hat{s}, \hat{y}_j^+ \rangle$ is bounded above by what is needed to get the desired increase δ in ξ_j:

$$\left\langle \hat{s}, \hat{y}_j^+ \right\rangle \leqslant \langle \hat{s}, \hat{y}_j \rangle + \delta \langle s, H_j s \rangle . \tag{5.20}$$

A lower bound is also imposed on $\langle \hat{s}, \hat{y}_i^+ \rangle$ to prevent it to become (too) negative:

$$\left\langle \hat{s}, \hat{y}_i^+ \right\rangle \geqslant -\left\langle \check{s}, \check{y}_i \right\rangle + \tau \langle s, H_i s \rangle , \tag{5.21}$$

where τ determines how negative ξ_i^+ may become. Obviously, we also impose that

$$\xi_j^+ > \xi_j. \tag{5.22}$$

The updated vectors y_i^+ and y_j^+ are then given by (5.15), and (5.17) or (5.18) with the maximal value of β satisfying the conditions (5.19) to (5.22) (if there are compatible).

Note that requiring v to be orthogonal to w can be a restrictive choice when M is sufficiently large (more precisely, when the intersection of lines $w + \mathbb{R}v$ and $\mathbb{R}\hat{s}$ lies inside the ball of radius M centred at the origin). We then turn back to (5.18) even if $\hat{s}$ and w are not collinear. This completes the push operation description.

Solving the feasible flow problem. — Now that we have introduced our flow network and discussed how we could transfer flow (that is curvature) amongst its nodes, we explain how we solve the feasible flow problem, in which the demand at each node is the slack in (5.14).

As explained in Subsection 1.4.2, this network is first transformed into a maximal flow problem between a source node and a sink node. At that stage, we could normally apply the push-relabel method (see also Subsection 1.4.2). However each arc capacity of our network depends on the values of the elemental gradient variations corresponding to the extremities of the arc. The

arc capacities are thus likely to vary along the shifting process. Our problem therefore departs from the classical flow problem in graph theory (where the capacities are fixed independently of demand). Moreover, the total flow $\sum \xi_i$ is not necessarily conserved after a modification of the flow (due to the definition (5.14)). As a consequence, the arc capacities and node demand need to be updated after each round of flow modifications, which in turn results in an implicit update of the total flow. The flow analogy is then used as a heuristic, in which the process used to define pushes, but without condition (5.20), is also used to initialize the arcs capacities and to update every arc adjacent after a push.

For the implementation of the push-relabel method, we chose the *lowest-label* strategy, because of its numerical performance on this problem, and because it starts by considering the deficient nodes (that have the lowest labels), which are, in our case, the only problematic ones. Some refinements of the method (see notably the *global relabelling* in Cherkassky and Goldberg, 1997, and the *gap relabelling* in Cherkassky, 1979, and Derigs and Meier, 1989) were also used in our numerical codes, based on the implementation of the *highest-label push-relabel* (HIPR) code of Andrew V. Goldberg.

Note that this procedure aims at solving the following underlying linear program:

$$\begin{aligned}
&\text{Find} && \{y_i\}_{i=1}^m && && (5.23a)\\
&\text{s.t.} && \textstyle\sum_{i=1}^m y_i = y && && (5.23b)\\
& && \xi_i(y_i) \geqslant \epsilon\bar{\xi} \quad \text{and} \quad \|y_i\| \leqslant M && \text{for } i = 1, \ldots, m, && (5.23c)\\
& && [y_i]_j = 0 && \text{for } i = 1, \ldots, m \text{ and } j \in \mathcal{I}_i^{\mathsf{c}}, && (5.23d)
\end{aligned}$$

Balanced partitioned BFGS update. — Practically, the curvature balancing process can stop prematurely with nodes still deficient, because the constraints (5.19) to (5.22) become incompatible and thus the arc capacities reduce to zero. Therefore, some additional heuristics were tested to accept or refuse the generated set of element gradient differences. The first one was to keep the new version of vector y_i only if the corresponding ξ_i was initially positive or was not decreased. The second one was to monitor each element secant equation and to update the element gradient difference y_i only if the corresponding i-th element secant equation was not deteriorated with respect to the initial set of vectors y_i. The best results were obtained by combining these heuristics: each y_i was updated only if both rules were satisfied. This obviously generates a relaxation of the summation condition (5.8), which is equivalent to relaxing the full secant equation.

The last algorithm (Algorithm 5.5) which we propose to test therefore consists in uniformly splitting the gradient difference y and then applying the elemental BFGS updates on the basis of the element gradient differences y_i balanced using the heuristics described above.

Algorithm 5.5 (Balanced partitioned BFGS update: BP-BFGS)

The current element Hessian approximation B_i, the step s, the corresponding gradient variation y, and a threshold $\kappa > 0$ are given.

Step 1. Decompose vector s into $\{s_i\}_{i=1}^m$ and split uniformly y into $\{y_i\}_{i=1}^m$.

Step 2. Apply the push-relabel heuristic on the curvature flow network described above, in order to balance the vectors y_i.

Step 3. For each element i, check for initial positivity or improvement of ξ_i, and non-deterioration of the i-th element secant equation. If these conditions fail, use the initial vector y_i.

Step 4. Perform the BFGS update (4.23) on each B_i for which $\langle s_i, y_i \rangle \geqslant \kappa \langle s_i, B_i s_i \rangle$.

5.5 Numerical experiments

We performed our numerical experiments inside the RMTR$_\infty$ method (see Section 3.2). More precisely, we used the Fortran 95 implementation written by Dimitri Tomanos (see notably Tomanos, 2009) with the parameter values advised by Gratton *et al.* (2010*b*). All codes were written in Fortran 95 and experiments were conducted on a 3.40 GHz Intel® Pentium® dual-core processor computer with 2 GB of RAM.

Our modifications to the RMTR$_\infty$ code intend to deal with approximate Hessians. So, instead of reevaluating the Hessian when needed, it was approximated or updated using one of the compared procedures:

LTS: the lower triangular substitution method (see Algorithm 5.1);

LTS-O: the lower triangular substitution method using optimal column groupings (see Subsection 5.1.2);

S-PSB: the sparse PSB update (see Algorithm 5.2);

PS-PSB: the partially separable PSB update (see Algorithm 5.3);

UP-BFGS: the partitioned BFGS update with the uniform splitting of the gradient differences (see Algorithm 5.4);

BP-BFGS: the partitioned BFGS update with the balanced splitting of the gradient differences (see Algorithm 5.5).

Note that the element gradients and Hessians need to be stored for the UP-BFGS and BP-BFGS choices, in contrast to the PS-PSB option, which uses them implicitly.

5.5.1 Practicalities

Except for the LTS and LTS-O methods, the initial Hessian at each level of the algorithm has to be estimated rather than evaluated; we choose to set every element Hessian B_i to the identity matrix. This initialization is implicit for the S-PSB and PS-PSB algorithms because the element Hessians are never stored individually.

In the PSB-like methods, the required accuracy on the solution λ was set to 10^{-6}. In the BP-BFGS algorithm, the bound M on the norms was set to $\|y\|$, ϵ to 0.1 and τ to 0.

Moreover, for the PSB-like algorithms, the i-th element Hessian B_i was not updated if the norm of the corresponding element step s_i was smaller than 10^{-6} times the norm of the step s. As nearly no information can be collected in the directions given by these s_i, this technique can be used with little effect except that of preventing bad conditioning of the linear system for the PSB-like updates. Similarly, we set κ to 10^{-6} for the partitioned BFGS algorithms.

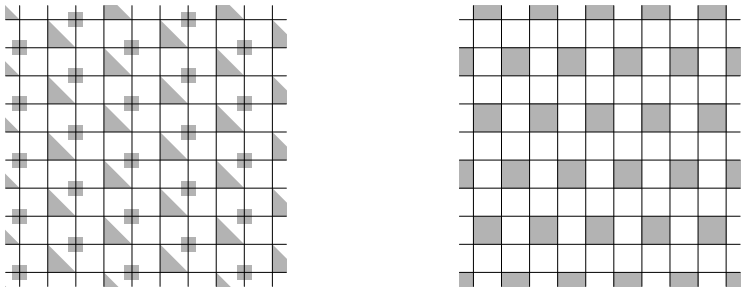
5.5.2 Test problems

We have considered the minimization problems listed in Table 5.1 with different sizes (that correspond to different choices of their finest level). A short description of these problems is provided in Subsection 3.2.3.

Covering molecules. — We here present the covering molecules used for these problems with the LTS-O algorithm, and which are given by Goldfarb and Toint (1984), except for problems `NCCO` and `NCCS`.

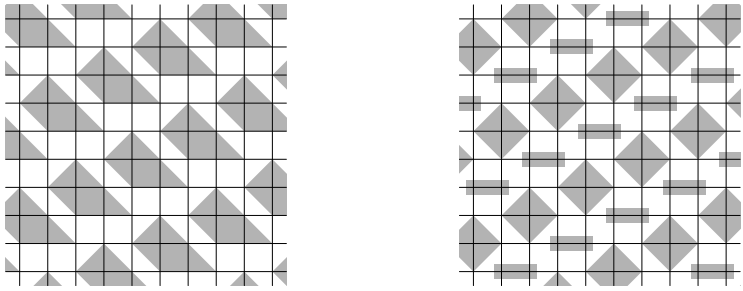
Problems `P2D`, `BRATU` and `MEMBR` arise from a 5-point finite difference Laplacian operator; this gives a 5-diagonal Hessian, whose cover is displayed in Figure 5.2. The Hessian of the 3-dimensional Laplacian problem `P3D` consists in 7 diagonals and an horizontal layer of its cover is represented in Figure 5.3a; the molecules are in fact tetrahedrons. In problems `DEPT`, `DPJB`, `DODC`, `MINS-SB`, `MINS-OB`, `MINS-DMSA` and `DSSC`, the Hessian has also 7 diagonals and its cover is displayed in Figure 5.3b.

For problems `IGNISC` and `MOREBV`, we have a 13-diagonal Hessian, whose cover is displayed in Figure 5.4a. Finally, while Goldfarb and Toint (1984) present no cover for the two problems `NCCS` and `NCCO`, we describe one in Figure 5.4. As these problems use two sets of variables, the cover is also defined on two layers, the first one corresponding to that of a 13-diagonal Hessian.



(a) (b)

Figure 5.3 — *Covers for a 3-dimensional 7-point Laplacian operator (horizontal layer)* (a)*, and a 2-dimensional 7-diagonal Hessian* (b)*.*



(a) (b)

Figure 5.4 — *Covers for a 13-diagonal Hessian* (a)*, and for* `NCCS`/`NNCO` *problems (first set of variables in* (a)*, second one in* (b)*).*

Problem name	Sizes		
P2D	261,121	1,046,529	4,190,209
P3D	29,791	250,047	
DEPT	261,121	1,046,529	4,190,209
DPJB	261,121	1,046,529	4,190,209
DODC	261,121	1,046,529	4,190,209
MINS-SB	65,025	261,121	1,046,529
MINS-OB	65,025	261,121	1,046,529
MINS-DMSA	65,025	261,121	1,046,529
IGNISC	261,121	1,046,529	
DSSC	261,121	1,046,529	4,190,209
BRATU	65,025	261,121	1,046,529
MEMBR	261,121	1,046,529	4,190,209
NCCS	7,938	32,258	130,050
NCCO	32,258	130,050	522,242
MOREBV	65,025	261,121	1,046,529

Table 5.1 — *Test problems for sparse and partially separable Hessian methods comparison.*

5.5.3 Results

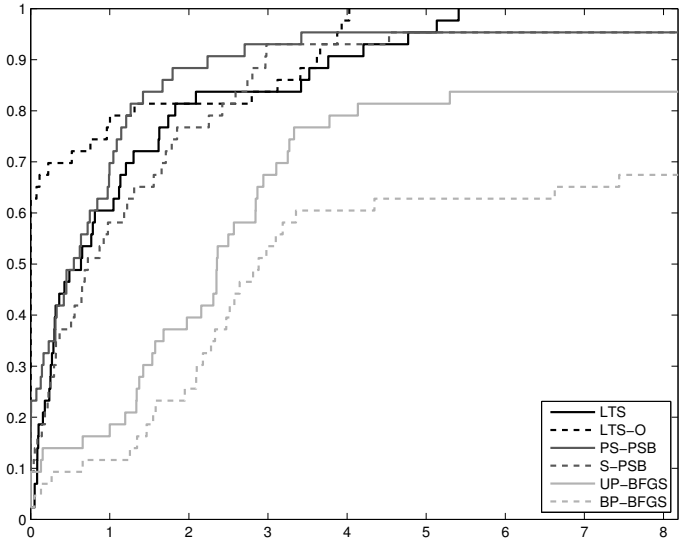
As function and gradient evaluations do not cost the same at each level, we defined an equivalent number of evaluations (see Gratton *et al.*, 2010*b*) given by

$$q = \sum_{\text{level } \ell} q_\ell \frac{n_\ell}{n}$$

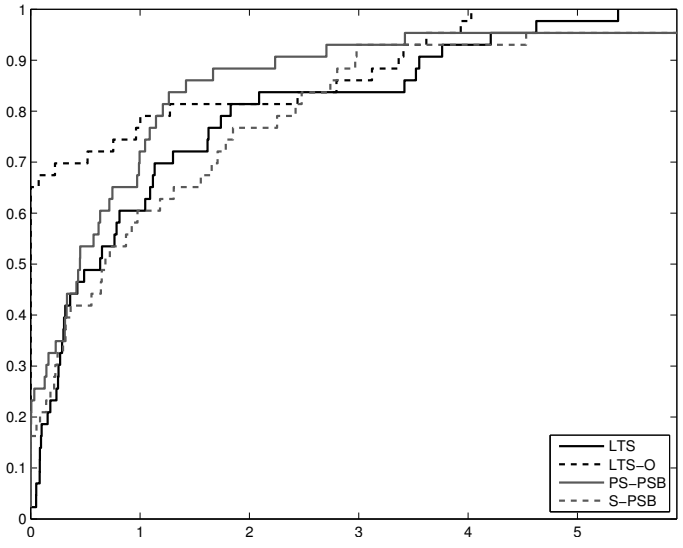
where q_ℓ and n_ℓ are respectively the number of evaluations and the size of the problem at level ℓ. The results of the numerical experiments are given in Section A.2 in appendix. For better readability, we display these results as performance profiles. In Figure 5.5, we take the number of function evaluations plus five times the number of gradient evaluations as comparison criterion; this ratio seems appropriate in view of the evaluation cost of a gradient by automatic differentiation (see Griewank, 1989). In Figures 5.6 and 5.7, we compare the CPU time and the equivalent number of Hessian updates, respectively.

The graphs show that the two LTS-based methods and the two PSB-type updates are clearly more efficient and robust than the partitioned BFGS updating procedures. The poor robustness of the latter methods is in part due to the iteration and time limit.

Comparing LTS and LTS-O, we observe that if an optimal column grouping is available, it is worth exploiting it, because it allows an important reduction of the number of gradient evaluations. A reduction of the CPU time is also

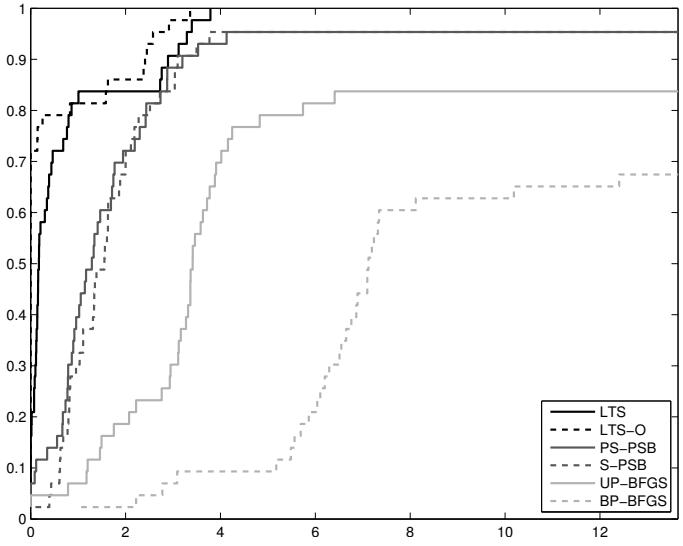


(a) *including all methods.*

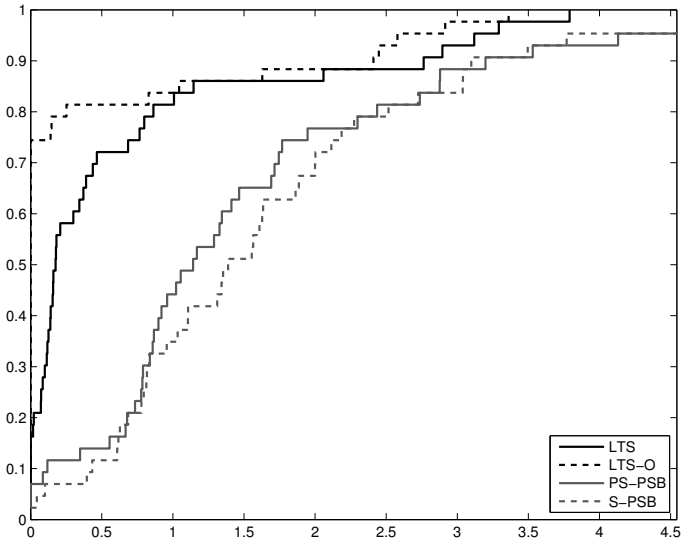


(b) *excluding the partitioned BFGS methods.*

Figure 5.5 — *Performance profiles based on the number of function evaluations plus five times the number of gradient evaluations.*

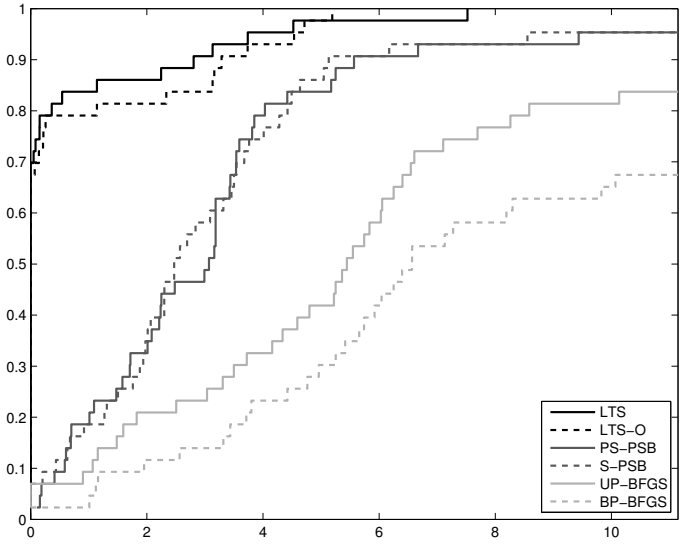


(a) *including all methods.*

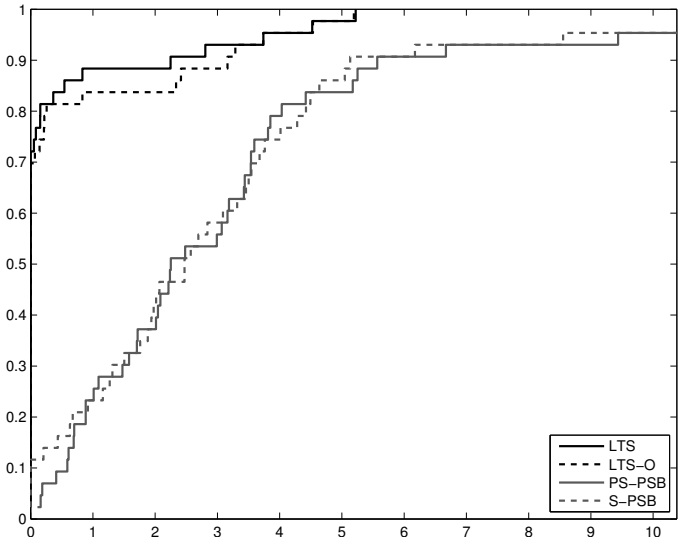


(b) *excluding the partitioned BFGS methods.*

Figure 5.6 — *Performance profiles based on the CPU time.*



(a) *including all methods.*



(b) *excluding the partitioned BFGS methods.*

Figure 5.7 — *Performance profiles based on the number of Hessian updates.*

observed, but it corresponds mainly to the smaller number of gradient evaluations. Indeed, as both methods give a relatively accurate approximation of the Hessian, the numbers of iterations requiring a Hessian update are quite similar.

In terms of robustness, the LTS-based methods and the PSB-type updates give similar results, with a small advantage for the LTS-based methods. Remember that the LTS-based methods require more gradient evaluations than the PSB-type updates at each iteration where the Hessian is (approximately) recomputed. This should be a disadvantage, but as shown in Figure 5.7, the better accuracy obtained with the LTS-based methods reduces the number of iterations where the Hessian needs to be recomputed in RMTR. We obtain similar results in terms of function and gradient evaluations (and in fact, better results for the LTS-O variant as it requires less gradient evaluations). If we now consider Figure 5.6, we observe that the LTS-based methods are faster than the PSB-type updates. This seems to correspond to the additional iterations for the PSB-type updates and to the fact that the LTS-based methods solve a sparse *triangular* linear system instead of a sparse general linear system. A small advantage is also observed for the partially separable PSB update with respect to the sparse one.

Regarding the two partitioned BFGS updating process, we note that, surprisingly, the unbalanced version appears a little more efficient and robust. Clearly, the balancing procedure is also too expensive to produce good results in CPU time. This initially interesting idea thus appears not to give the associated advantages. In fact, the RMTR method seems to deal pretty well with directions of negative curvature, and the quest for sparse positive definite secant Hessian approximation may just be a lure. Ultimately, it may be unreasonable to fool the trust-region method with positive definite Hessian approximation when the objective function is highly nonconvex, and to therefore produce a model whose behaviour may highly differ from that of the objective function.

We finally note that none of our examples required more than 15 gradient differences with the LTS-based algorithms to evaluate the complete Hessian. Since the same Hessian value is often used for more than a single iteration, the algorithm typically requires less than 5 gradient differences per iteration. If techniques using Hessian-times-vector products were used instead (and ignoring the need of knowing individual Hessian entries for smoothing), they would occur in the iterative solution of a linear system whose solution is the approximate Newton direction. Because the cost of such a product is essentially that of a gradient difference, a competitive computation of the truncated Newton steps would be limited to 5 products per iteration, which puts a severe limit on the accuracy of these steps.

Our numerical comparison thus indicates that the LTS method is a good choice in terms of effectiveness and robustness, all the more so as an optimal column grouping is known. This method has therefore been added in the GALAHAD library, which will be discussed in Chapter 8 devoted to software.

5.6 Perspectives

Disregarding whether a positive definite update should be sough, determining an appropriate splitting of the gradient difference y remains an open question. In fact, the problem (5.23) could be (approximately) solved by general optimization techniques. It holds furthermore two artificial constants ϵ and M that it may be wiser to eliminate. For instance, we might consider instead the quadratic program

$$\begin{aligned}
\min_{\{y_i\}} \quad & \tfrac{1}{2}\textstyle\sum_{i=1}^{m}\|y_i\|^2 && && \text{(5.24a)}\\
\text{s.t.} \quad & \textstyle\sum_{i=1}^{m} y_i = y, && && \text{(5.24b)}\\
& \xi_i(y_i) \geqslant \epsilon\bar{\xi} && \text{for } i = 1,\dots,m, && \text{(5.24c)}\\
& [y_i]_j = 0 && \text{for } i = 1,\dots,m \text{ and } j \in \mathcal{I}_i^{\mathsf{c}}, && \text{(5.24d)}
\end{aligned}$$

where ϵ is still set to a small positive value. Enforcing the positivity of $\xi_i(y_i)$ might also be obtained through penalization, but thereby adding penalization parameters, which should also be tuned. Finally, we must keep in mind that, in any case, the Hessian approximation cost should remain reasonable, especially with respect to that of finite-difference approximations, which are relatively accurate.

Chapter 6

Limited-memory methods for Hessian approximation

Limited-memory methods are designed as an intermediary between conjugate gradient methods and quasi-Newton methods. The former indeed keep information on the objective function curvature in a single vector, while the later use a complete matrix at this purpose. The memory devoted to this curvature information will therefore be limited (hence the methods' name) to a small number of vectors. Methods including this limitation are thus suitable for large-scale problems (the required memory can even be controlled by the user) and are hoped to converge faster than conjugate gradient methods since more information on the function curvature is available. They originated with the works of Perry (1977) and Shanno (1978*a*), and were subsequently developed and analyzed by Buckley (1978*a*,*b*), Nazareth (1979), Nocedal (1980), Shanno (1978*b*), Gill and Murray (1979), and Buckley and LeNir (1983). Compared to other techniques studied in the previous chapter, they also remain relatively simple, since the Hessian sparsity pattern or the partial separability structure are not required.

We begin our discussion on these methods with the limited-memory BFGS (L-BFGS) method of Nocedal (1980) (in Section 6.1), since it turns out to be particularly efficient amongst limited-memory methods (see in particular Liu and Nocedal, 1989, and Gilbert and Lemaréchal, 1989). It is based on maintaining a small set (of fixed size) containing secant pairs that approximately represent the function curvature, and are thus used to compute an approximate Hessian with the BFGS formula. These secant pairs are taken as the past iteration steps and the corresponding gradient variations. Gratton and Toint (2010) however propose a new framework for selecting these secant pairs (Section 6.2). In particular, they suggest that the multigrid hierarchy can be used for their design. In Section 6.3, we study this possibility and try to characterize the

new pairs. We then evaluate them numerically in Section 6.4 for the design of both a Hessian approximation in quasi-Newton methods and a preconditioner for conjugate gradient methods. In these experiments, we however simply use the new secant pairs inside the L-BFGS formula of Nocedal (1980), which appears as a black box. It is therefore the intent of the last section of this chapter (Section 6.6) to consider some other strategies to use the curvature information contained in these pairs to define a Hessian approximation.

Contrary to the previous (and next) chapters, we do not use the RMTR method in the present chapter. The multilevel framework is however used, but in another way.

6.1 L-BFGS method

In the L-BFGS method, the Hessian matrix is assembled at each iteration as the sum of finitely many low-rank corrections, each involving a secant pair. We assume that at most m pairs may be stored. So we consider, at each iteration k, a set $\mathcal{P}_k$ containing $m_k \leqslant m$ pairs $(s_{k,j}, y_{k,j})$, for $j = 1, \ldots, m_k$, that provide the available curvature information on the objective function. As already mentioned, it is common to approximate the inverse Hessian rather than the Hessian. Following this trend, the L-BFGS method chooses at each iteration some initial matrix $H_{k,0}$, and then updates it with the BFGS formula using successively the secant pairs contained in $\mathcal{P}_k$. Hence the new approximated inverse Hessian H_{k+1} is defined as H_{k,m_k}, the last element from the recursion

$$H_{k,j} = \left(I - \rho_{k,j} s_{k,j} y_{k,j}^T\right) H_{k,j-1} \left(I - \rho_{k,j} y_{k,j} s_{k,j}^T\right) + \rho_{k,j} s_{k,j} s_{k,j}^T \tag{6.1}$$

with $\rho_{k,j} = \langle s_{k,j}, y_{k,j} \rangle^{-1}$, for $j = 1, \ldots, m_k$. The L-BFGS method of Nocedal (1980) uses the past iteration steps to define the secant pairs. More precisely, it sets $m_k = \min(k, m)$, and takes the secant pairs $s_{k,j} = x_{k-m_k+j+1} - x_{k-m_k+j}$ and $y_{k,j} = g_{k-m_k+j+1} - g_{k-m_k+j}$ for $j = 1, \ldots, m_k$.

As mentioned by many authors, the choice of the initialization matrix $H_{k,0}$ is quite important to get a well-scaled search direction. A common choice is to consider particular multiples of the identity matrix, and especially

$$H_{k,0} = \gamma_k I \quad \text{with} \quad \gamma_k = \frac{\langle s_{k,m_k}, y_{k,m_k} \rangle}{\|y_{k,m_k}\|^2}.$$

Besides, we are only interested in the product of H_{k+1} by some vector in many cases (for instance to define the quasi-Newton direction in linesearch methods, or in the solution of the trust-region subproblem using the truncated conjugate gradient method), and thus need not to explicitly construct the matrix H_{k+1}. Nocedal (1980) shows that this product can be computed using instead the (matrix-free) two-loop recursion reported in Algorithm 6.1.

Algorithm 6.1 (L-BFGS: two-loop recursion)

A vector g by which H_{k+1} must be multiplied is given.

Step 1. Set $q = g$.

Step 2. For j decreasing from m_k to 1:

(a) Compute $\alpha_j = \rho_{k,j} \langle s_{k,j}, q \rangle$.

(b) Update $q \leftarrow q - \alpha_j y_{k,j}$.

Step 3. Set $p = H_{k,0} g$.

Step 4. For j increasing from 1 to m_k:

(a) Compute $\beta = \rho_{k,j} \langle y_{k,j}, p \rangle$.

(b) Update $p \leftarrow p + (\alpha_j - \beta) s_{k,j}$.

Step 5. Return with the product $p = H_{k+1} g$.

6.2 Approximate invariant subspaces

Gratton and Toint (2010) start with the observation that the secant equation (4.12) is automatically fulfilled by the mean Hessian $\bar{B}_k$ on the segment $[x_k, x_{k+1}]$. This somehow constitutes a definition of what is a secant pair (or a secant equation) at iteration k: a pair (s, y) that satisfies

$$\bar{B}_k s = y. \tag{6.2}$$

It may then be reasonable to use the curvature information that it provides in an (inverse) Hessian approximation at iteration $k + 1$.

Assume now that we know a collection $\{\mathcal{S}_i\}_{i=1}^r$ of invariant subspaces of $\bar{B}_k$, and the orthogonal projectors S_i onto these spaces $\mathcal{S}_i$. Since these projectors share a common system of eigenvectors with $\bar{B}_k$, they commute, that is $\bar{B}_k S_i = S_i \bar{B}_k$. Then, given an existing secant pair (s, y) at iteration k, we then obtain

$$\bar{B}_k S_i s = S_i \bar{B}_k s = S_i y \tag{6.3}$$

thereby yielding a new secant equation with the pair $(S_i s, S_i y)$. Repeating the procedure for $i = 1, \ldots, r$, we therefore obtain r additional secant equations (in addition to the original one). Since the eigenvectors of a matrix and its inverse are identical, the same result holds if we work with the inverse Hessian. So the pair $(S_i s, S_i y)$ also constitutes a valid secant pair for the inverse secant equation.

If the subspaces $\mathcal{S}_i$ are only approximately invariant, or if the operators S_i are only approximately equal to projectors onto these subspaces, then the secant equations stop being exact but can be expected to hold approximately. We therefore refer to secant equations of the type (6.3) as approximate, in contrast to the exact equation (4.12). In this case, Gratton and Toint (2010) provide a bound on the backward error on the secant equation. Consider the decomposition $S_i = Q_i D_i Q_i^T$, where the columns of Q_i form an orthonormal basis of $\mathcal{S}_i$, and D_i is diagonal of dimension $\dim(\mathcal{S}_i)$. We define

$$G_i = Q_i^T \bar{B}_k Q_i \quad \text{and} \quad F_i = (Q_i^{\mathsf{c}})^T \bar{B}_k Q_i^{\mathsf{c}} \tag{6.4}$$

where Q_i^{c} is chosen such that the matrix $[Q_i \,|\, Q_i^{\mathsf{c}}]$ is orthogonal, and similarly decompose $s = Q_i \hat{s} + Q_i^{\mathsf{c}} \check{s}$. Let $E_i \in \mathbb{R}^{n \times n}$ be any symmetric perturbation such that $(\bar{B}_k + E_i) S_i s = S_i y$. Then

$$\frac{\|E_i\|}{\|\bar{B}_k\|} \leqslant \frac{\|G_i D_i - D_i G_i\|}{\sigma_{\min}(D_i)\|\bar{B}_k\|} + \kappa(D_i) \frac{\|F_i\|}{\|\bar{B}_k\|} \frac{\|s\|}{\|\hat{s}\|}, \tag{6.5}$$

where $\sigma_{\min}(D_i)$ and $\kappa(D_i)$ are the smallest singular value and condition number of D_i, respectively. The relative perturbation to $\bar{B}_k$ should therefore be small when S_i is an approximate projector (in which case D_i is close to the identity matrix and of modest conditioning), and if the orthogonal complement term F_i is small compared to $\|\bar{B}_k\|$ (which is expected if $\mathcal{S}_i$ is approximately an invariant subspace of $\bar{B}_k$) together with $\|\hat{s}\|$ being non-marginal with respect to $\|s\|$. This last condition is also acceptable, since it would not be interesting to exploit the curvature information along a projected step $\hat{s}$ which is vanishingly small compared to the complete step s, because rounding errors would then make this information unreliable.

6.3 Filtering secant pairs on the multilevel hierarchy

In particular, Gratton and Toint (2010) propose to take advantage of the multilevel hierarchy of problem (3.20) to define such approximate secant pairs. Indeed, the efficiency of multigrid methods relies on the fact that the image of any smooth mode by the matrix A (defining the linear system) is still (approximately) smooth, allowing a (rather) uncoupled work on different grids. Extrapolating this observation to nonlinear optimization leads to consider as potential invariant subspaces $\mathcal{S}_i$, the sets of smooth modes. We then remember that the range of the common prolongation operators P_i consists mostly of smooth modes (even if they are often contaminated by a small amount of oscillatory modes). Gratton and Toint (2010) therefore suggest to use the range of the combined prolongations $P_r \cdots P_{i+1}$ as the approximate invariant subspaces $\mathcal{S}_i$. Then, considering that each operator S_i needs to be applied twice

per iteration, their cost should be limited. So, they suggest to use

$$S_i = P_r \cdots P_{i+1} R_{i+1} \cdots R_r \qquad (i = 0, \ldots, r-1), \tag{6.6}$$

The transfer operators are indeed typically cheap to apply: the prolongation is for instance often chosen as the linear interpolation operator and the restriction as some multiple of its transpose, sometimes called the full-weighting operator. Computing the orthogonal projectors onto the subspaces $\mathcal{S}_i$ would be too expensive, and preliminary tests even indicate worse results than with the simpler expression (6.6).

The resulting approximate secant pairs $(S_i s, S_i y)$ are filtered versions of the secant pair (s, y) whose oscillatory components were mainly removed, bringing hence to the algorithm a wider range of curvature information on the objective function, and potentially helping it to reduce faster the smooth modes of the error. We therefore refer to these pairs as *smoothed pairs*, and to the particular vectors $S_i s$ as *smoothed steps*. We examine them in more detail in Section 6.5, after having explained the numerical elements in which they will be used.

Other potential filter operators. — Consider for the ease of the description a two-level problem with P and restriction operator R. Instead of using the combination PR to smooth the secant pairs, we would like to try two other operators. The first one is the true orthogonal projection on the range of P, that is $P(RP)^{-1}R$. The second one only differs from the first by the inner product used in the projection; this time, we use the Hessian-inner product, yielding the projector $P(RHP)^{-1}RH$. Preliminary results (using two of the strategies described in Section 6.4) are displayed in Tables 6.1 and 6.2. We observe differences between the results according to the used linesearch algorithms, but nonetheless no improvements are obtained with these two variants.

	P2D, $n = 127^2$, $m = 10$			MINS-SB, $n = 63^2$, $m = 7$	
	$\delta = 10^{-4}$	$\delta = 10^{-1}$	exact	$\delta = 10^{-4}$	$\delta = 10^{-1}$
L-BFGS	296	289	215	207	206
PR	139	124	197	109	110
$P(RP)^{-1}R$	152	130	363	147	116
$P(RHP)^{-1}RH$	191	184	196	138	159

Table 6.1 — *Number of iterations for the L-BFGS method and the* Local *multisecant method with different "filtering" operators. Either the Dennis-Schnabel linesearch with $\delta = 10^{-4}$ or $\delta = 10^{-1}$, or an exact linesearch was used.*

	P2D, $n = 127^2$, $r = 5$			MINS-SB, $n = 63^2$, $r = 4$	
	$\delta = 10^{-4}$	$\delta = 10^{-1}$	exact	$\delta = 10^{-4}$	$\delta = 10^{-1}$
L-BFGS	320	313	215	243	250
PR	153	138	277	173	160
$P(RP)^{-1}R$	301	357	2,303	214	154
$P(RHP)^{-1}RH$	322	347	333	287	307

Table 6.2 — *Number of iterations for the L-BFGS method and the* MLess *multisecant method (both with $m = r+1$) with different "filtering" operators. Either the Dennis-Schnabel linesearch with $\delta = 10^{-4}$ or $\delta = 10^{-1}$, or an exact linesearch was used.*

6.4 Numerical experiments

We now present our experiments with the approximate secant pairs built from the multilevel hierarchy of the problem under consideration. We explain the method and variants that were compared (Subsection 6.4.1), then mention the problems that we used (Subsection 6.4.2), and finally the results (Subsection 6.4.3)

6.4.1 Tested methods

In their experiments, Gratton and Toint (2010) considered a linesearch method with quasi-Newton search directions obtained with the L-BFGS Hessian approximation. Here, we consider in addition preconditioned nonlinear conjugate gradient methods, remembering the strong connection between these methods. For instance, Nazareth (1979) interestingly remarks that the quasi-Newton BFGS method with exact linesearch may be interpreted as a conjugate gradient method if we take the BFGS preconditioner H_k given by (4.24) and the (preconditioned) Hestenes-Stiefel conjugacy factor

$$\beta_k^{\mathrm{HS}} = \frac{g_{k+1}^T H_k y_k}{d_k^T y_k}$$

(compare with (2.54), on page 38). Observe also that choosing a zero conjugacy factor β_k turns a preconditioned nonlinear conjugate gradient method into a quasi-Newton linesearch method.

A classical strategy to choose a preconditioner for nonlinear conjugate gradient methods consists in fact in taking an approximation to $\nabla^2 f(x_*)^{-1}$ coming from a quasi-Newton formula. We therefore use the L-BFGS formula to define the Hessian approximation in the quasi-Newton search direction as well as the preconditioner of the nonlinear conjugate gradient method.

The algorithmic variants under consideration in these numerical tests therefore differ in the step length selection procedure, in the conjugacy factor used to

define the search direction, and in the secant pairs used in the L-BFGS update. We now present the considered alternatives for these characteristics.

Step length selection. — Whether the quasi-Newton direction or the conjugate gradient direction is taken, a linesearch is performed at each iteration. We first consider the step length selection procedure of Dennis and Schnabel (1983) (Algorithm 2.3), since it was used by Gratton and Toint (2010) in their experiments. We also add to the comparison the method from Hager and Zhang (2005) (Algorithm 2.7), since it appears to be the most efficient for nonlinear conjugate gradient methods to date (see Hager and Zhang, 2006*a*).

Conjugacy factor. — We consider three choices for the conjugacy factor β_k. The first one is

$$\beta_k^{\mathrm{QN}} = 0, \tag{6.7}$$

corresponding to the quasi-Newton methods, since the search direction given by (2.49) is then the same as in (2.15) with $M^{-1} = H_{k+1}$. The next two choices are advised by Hager and Zhang (2005) as the best choices currently known. Hence, our second choice is the preconditioned Hager-Zhang factor

$$\beta_k^{\mathrm{HZ}} = \left(M_k^{-1} y_k - 2 d_k \frac{y_k^T M_k^{-1} y_k}{d_k^T y_k} \right)^T \frac{g_{k+1}}{d_k^T y_k}, \tag{6.8}$$

(compare with (2.60), on page 39), and our third choice is the preconditioned hybrid Dai-Yuan-Hestenes-Stiefel factor

$$\beta_k^{\mathrm{DYHS}} = \max \left[0, \min \left[\frac{y_k^T M_k^{-1} g_{k+1}}{d_k^T y_k}, \frac{g_k^T M_k^{-1} g_{k+1}}{d_k^T y_k} \right] \right], \tag{6.9}$$

(compare again with (2.59), on page 39). In every case, the preconditioner M_k^{-1} is chosen as the Hessian approximation H_{k+1} produced by the L-BFGS formula (Algorithm 6.1).

Secant pairs selection. — At iteration k, the set $\mathcal{P}_k$ used for the L-BGFS approximation may contain *exact* pairs of the form (s_ℓ, y_ℓ) and *smoothed* pairs of the form $(S_i s_\ell, S_i y_\ell)$ (for some $\ell \leqslant k$ and $i < r$). Different variants were proposed by Gratton and Toint (2010) to select the pairs kept in memory:

L-BFGS: this strategy only uses the m_k last exact secant pairs (that is those used in the original L-BFGS method by Nocedal, 1980).

Full: this strategy uses the m last generated pairs, making no difference between smoothed and exact pairs.

Local: this strategy uses only the $\min(r, m-1)$ smoothed pairs from the current iteration, in addition to past and current exact pairs.

Mless: this (*memory less*) strategy uses only (smoothed and exact) pairs from the current iteration.

These strategies are illustrated in Figure 6.1.

Secant pairs ordering. — Each exact pair is always integrated in the L-BFGS update after its smoothed versions. However, we still may choose the order of the smoothed pairs in the L-BFGS formula:

Coarse first: this strategy first integrates the pairs that have been smoothed on the coarser levels (that is integrating pairs $(S_i s_k, S_i y_k)$ with index i running from 0 to $r-1$).

Fine first: this strategy first integrates the pairs that have been smoothed on the finer levels (that is integrating pairs $(S_i s_k, S_i y_k)$ with index i running from $r-1$ to 0).

We illustrate these strategies in Figure 6.2, when the Full strategy is used to select the pairs.

Collinearity and Curvature Control. — Gratton and Toint (2010) moreover propose to control the collinearity of the smoothed pairs with respect to the original exact pair; the pairs that do not satisfy the condition

$$| \langle S_i s_k, s \rangle | \leqslant \tau \|S_i s_k\|_2 \|s_k\|_2, \tag{6.10}$$

for some $\tau \in (0, 1]$, are thus discarded. Additionally, we enforce the positivity constraint (4.22) by ignoring pairs that do not satisfy

$$| \langle S_i s_k, S_i y_k \rangle | \leqslant \mu \langle s_k, y_k \rangle , \tag{6.11}$$

for parameter $\mu \in (0, 1)$. We test the values 0.999 and 1.0 for the threshold τ, and set μ to 10^{-6}.

Starting point and stopping criterion. — We choose the starting point as $[x_0]_j = 0.5$ and consider that the algorithm has converged as soon as $\|g_k\|_\infty \leqslant 10^{-5}$.

Details on the code. — All codes are written in Fortran 95. The tests were performed on a bi-processor Intel® Xeon® X5482 computer (4 cores, 3.20 GHz) with 64 GB of RAM.

We use the CG_DESCENT code (version 3.0) of Hager and Zhang (2006*a*) as a framework for our numerical experiments. Our modifications concerns the alternative use of the Dennis-Schnabel linesearch, the preconditioning with the L-BFGS formula, and the alternative definitions of the conjugacy factor β_k. Our implementation of the Dennis-Schnabel linesearch is a translation of the MATLAB® code of Gratton and Toint (2010) to Fortran 95.

ℓ	$k-6$	$k-5$	$k-4$	$k-3$	$k-2$	$k-1$	k
(s_ℓ, y_ℓ)	$*$	$*$	$*$	$*$	$*$	$*$	$*$
$(S_2 s_\ell, S_2 y_\ell)$							
$(S_1 s_\ell, S_1 y_\ell)$							
$(S_0 s_\ell, S_0 y_\ell)$							

(a) *L-BFGS strategy.*

ℓ	$k-6$	$k-5$	$k-4$	$k-3$	$k-2$	$k-1$	k
(s_ℓ, y_ℓ)						$*$	$*$
$(S_2 s_\ell, S_0 y_\ell)$						?	$*$
$(S_1 s_\ell, S_0 y_\ell)$						$*$	$*$
$(S_0 s_\ell, S_0 y_\ell)$						?	$*$

(b) *Full strategy; the pair $(S_2 s_{k-1}, S_2 y_{k-1})$ or $(S_0 s_{k-1}, S_0 y_{k-1})$ is selected depending on the pairs ordering.*

ℓ	$k-6$	$k-5$	$k-4$	$k-3$	$k-2$	$k-1$	k
(s_ℓ, y_ℓ)				$*$	$*$	$*$	$*$
$(S_2 s_\ell, S_2 y_\ell)$							$*$
$(S_1 s_\ell, S_1 y_\ell)$							$*$
$(S_0 s_\ell, S_0 y_\ell)$							$*$

(c) *Local strategy.*

ℓ	$k-6$	$k-5$	$k-4$	$k-3$	$k-2$	$k-1$	k
(s_ℓ, y_ℓ)							$*$
$(S_2 s_\ell, S_2 y_\ell)$							$*$
$(S_1 s_\ell, S_1 y_\ell)$							$*$
$(S_0 s_\ell, S_0 y_\ell)$							$*$

(d) *Mless strategy.*

Figure 6.1 — *Illustrations of the secant pairs selection strategies when the memory limit is $m = 7$ and the number of coarse levels is $r = 3$. The stars indicate which pairs are included in $\mathcal{P}_k$.*

ℓ	$k-1$	k
(s_ℓ, y_ℓ)	3	7
$(S_2 s_\ell, S_2 y_\ell)$	2	6
$(S_1 s_\ell, S_1 y_\ell)$	1	5
$(S_0 s_\ell, S_0 y_\ell)$		4

(a) *Coarse first strategy.*

ℓ	$k-1$	k
(s_ℓ, y_ℓ)	3	7
$(S_2 s_\ell, S_2 y_\ell)$		4
$(S_1 s_\ell, S_1 y_\ell)$	1	5
$(S_0 s_\ell, S_0 y_\ell)$	2	6

(b) *Fine first strategy.*

Figure 6.2 — *Illustrations of the secant pairs ordering strategies when the* Full *strategy is used, the memory limit is $m = 7$, and the number of coarse levels is $r = 3$. We indicate in the arrays the numbering index j of the pairs in $\mathcal{P}_k$.*

6.4.2 Test problems

For our numerical experiments, we consider the unconstrained optimization problems provided by Gratton *et al.* (2010*b*). As before, we indicate in Table 6.3 the level and size at which they were considered, and refer to Subsection 3.2.3 for a description.

Name	Level	Size
DNT	8	511
P2D	8	261,121
P3D	5	250,047
DEPT	8	261,121
DODC	8	261,121
MINS-SB	8	261,121
MINS-OB	8	261,121
MINS-DMSA	8	261,121
IGNISC	8	261,121
DSSC	8	261,121
BRATU	8	261,121
NCCS	7	130,050
NCCO	7	130,050
MOREBV	8	261,121

Table 6.3 — *Test problems for the L-BFGS variants.*

6.4.3 Results

We ran a large number (78) of possible combinations of the variants described in Subsection 6.4.1 on our set of 14 test problems and report all results of the 1,092 runs on comet-shape graphs representing a measure of the effort spent in

function/gradient evaluations vs. iterations number. More precisely, we have first scaled, separately for each test problem, on the one hand, the number of function evaluations plus five times the number of gradient evaluations, and on the other hand, the iterations number, by dividing them by the best obtained for this problem by all algorithmic variants. We then plotted the averages of these scaled measures on all test problems for each algorithmic variant separately, after removing the variants that fails on at least one problem (the number of such variants is given in legend).

In the first of these plots (Figure 6.3), we have used squares for the variants where the quasi-Newton search direction is chosen, stars for the variants where the conjugate gradient search direction with the conjugacy factor β_k^{DYHS} is chosen, and triangles for the variants where the conjugate gradient search direction with the conjugacy factor β_k^{HZ} is chosen. We note a substantial spread

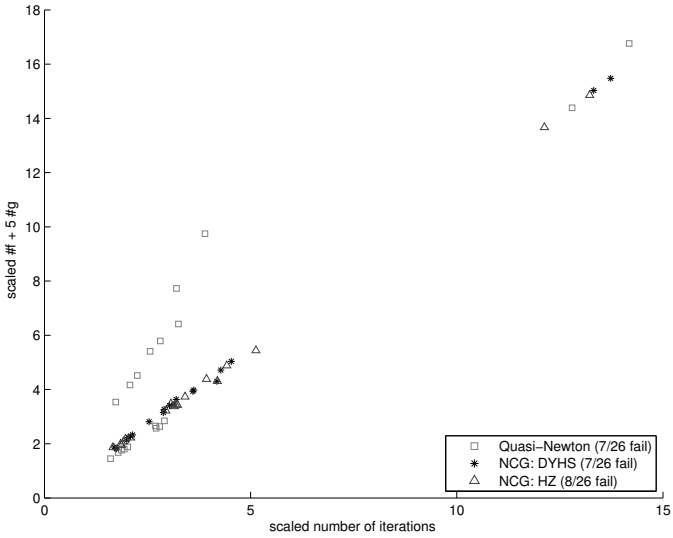


Figure 6.3 — *Comet-shape graph comparing the choice of conjugacy factor.*

of the results, with some options being up to 15 times worse than others. The worst cases (in the top right corner) correspond to combination of the Hager-Zhang linesearch, the Mless strategy and the collinearity threshold set to 0.999. Among the other variants, we may also observe a second set of variants which requires (in average) twice the number of function/gradient evaluations. These correspond to the use of the Hager-Zhang linesearch inside the quasi-Newton method. The third set of variants gather results from the quasi-Newton method (with the Dennis-Schnabel linesearch) and nonlinear conjugate gradi-

ent method, with in average, better results for the former methods than for the latter.

We next compare the effect of the linesearch choice in Figure 6.4. In that picture, squares have been used for the variants using the Hager-Zhang linesearch, and stars for the variants using the Dennis-Schnabel linesearch. We

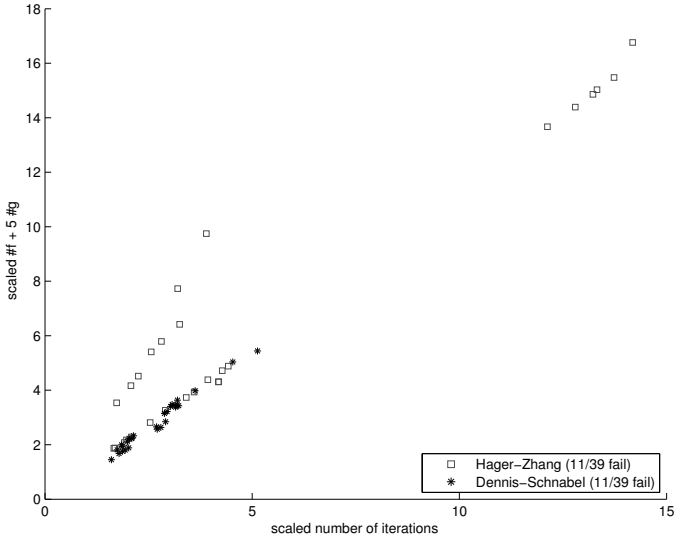


Figure 6.4 — *Comet-shape graph comparing the choice of linesearch.*

observe that the Hager-Zhang linesearch do not improve the performance, especially inside the quasi-Newton method, where the number of function and gradient evaluations per iterations is doubled in average. We would therefore advise the Dennis-Schnabel linesearch since it is also simpler.

Figure 6.5 compares the variants performance on the basis of the pairs selection strategy. This time, squares have been used for the variants using the L-BFGS strategy, stars for the Full strategy, triangles for the Local strategy, and crosses for the Mless strategy. We observe that the L-BFGS strategies are in a factor 4, in average, from the best ones. The Local and Mless strategies give best results. We recommend the Local strategy since the number of pairs used by the Mless strategy is limited by the number of levels, and thus prevent the full use of the available memory capacity.

We then compare the effect of the integration order of the smoothed pairs inside the L-BFGS update. In Figure 6.6, we have thus used squares for the variants that integrate first the pairs smoothed at the coarser levels (Coarse First strategy), and stars for the variants that integrate first the pairs smoothed at the finer levels (Fine First strategy). It is unclear which strategy is the more

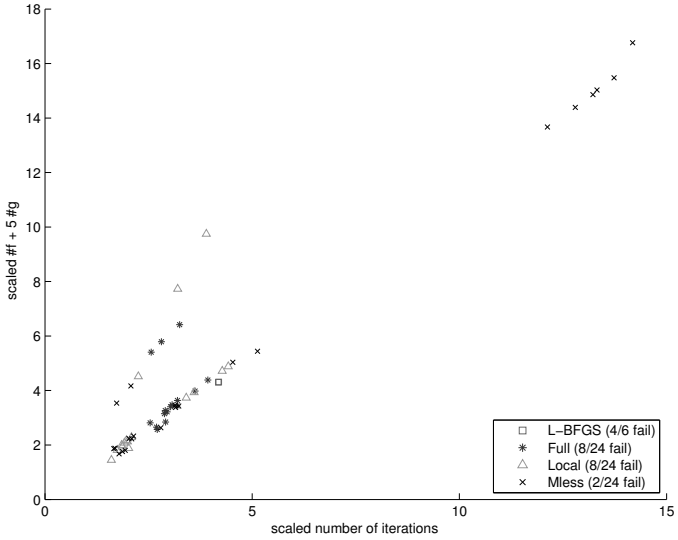


Figure 6.5 — *Comet-shape graph comparing the choice of pairs selection.*

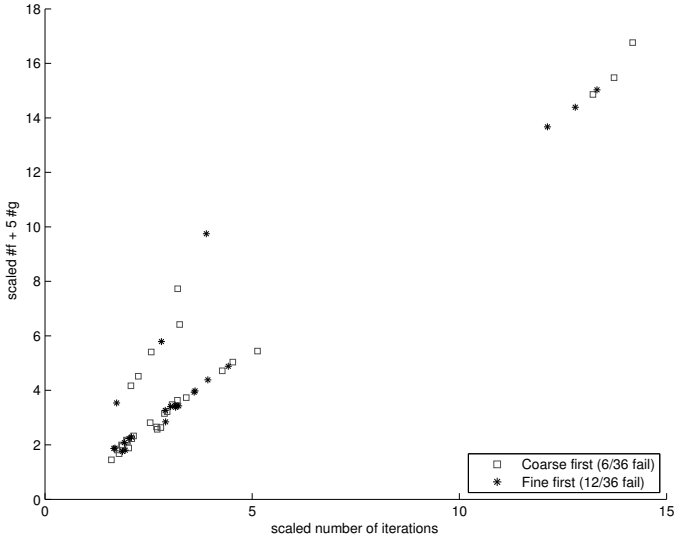


Figure 6.6 — *Comet-shape graph comparing the choice of pairs order.*

efficient. Nevertheless, the variants using the Coarse First strategy encounter less failures.

Finally, we consider the interest to control the collinearity of the smoothed pairs with respect to the exact secant pair from which they were generated. In Figure 6.7, we have thus used squares to represent the variants that do not control this collinearity (setting the threshold τ to 1.0), and stars to represent the variants that control this collinearity, with a threshold τ set to 0.999. The

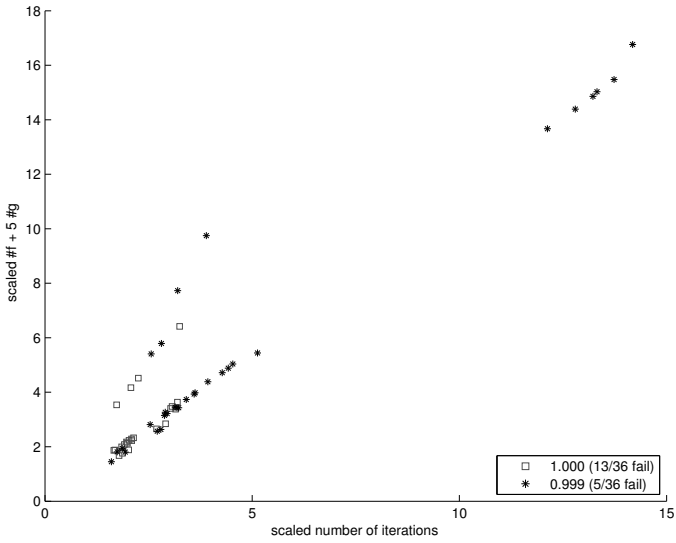


Figure 6.7 — *Comet-shape graph comparing the choice of pairs collinearity control.*

more efficient variant is again unclear, but the collinearity control seems to improve the robustness.

To summarize our conclusions so far, we may first attempt to distinguish the best variant *not* using smoothed secant pairs, and then compare the strategies using the smoothed pairs with this selected contender, and again look for the best of the set. These two steps are illustrated by Figures 6.8 and 6.9, where we have restricted ourselves to considering performance in terms of function and gradient evaluations (the performance profiles for the number of iterations are similar).

The first of these figures illustrates our findings well: the best method in our tests appears to be that using either the Dai-Yuan-Hestenes-Stiefel or the Hager-Zhang formula for deriving the search direction, coupled with the Hager-Zhang linesearch. Interestingly, if one restricts one's attention to quasi-Newton methods ($\beta_k = 0$), then the Dennis-Schnabel linesearch seems to dominate Hager-Zhang's by a substantial margin on our examples.

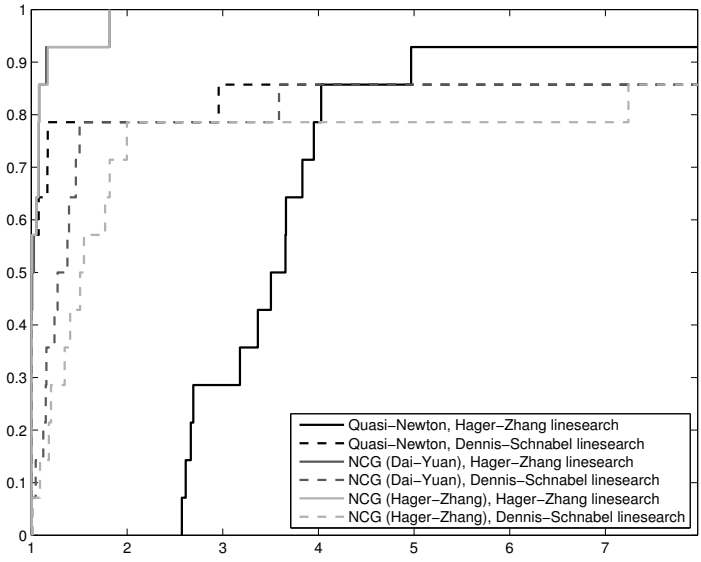


Figure 6.8 — *Performance profile for the variants not using smoothed secant pairs.*

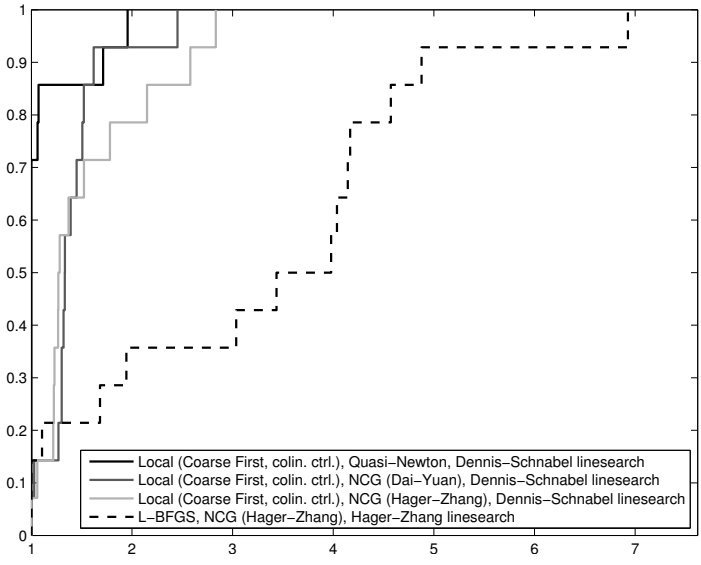


Figure 6.9 — *Performance profile for the variants using* Local *smoothed secant pairs against the best one not using them.*

The second performance profile also stresses our earlier conclusions: the use of smoothed secant pairs (in their Local flavour, which we already selected as best above) is clearly beneficial when possible. Indeed, all variants using them substantially dominate the best variant which does not. Amongst them the best variant is the quasi-Newton method using Local pairs, starting from the coarsest, together with collinearity control and Dennis-Schnabel linesearch. The gap between this variant and the second best (the conjugate gradient variant using the Dai-Yuan-Hestenes-Stiefel formula) is significant, although not as wide as that separating the variants using smoothed secant pairs from their best contender.

6.5 Smoothed pairs characterization

We first examine the smoothed pairs from the perspective of accuracy (Subsection 6.5.1), and then from that of novelty (Subsection 6.5.2).

6.5.1 Smoothed secant pairs accuracy

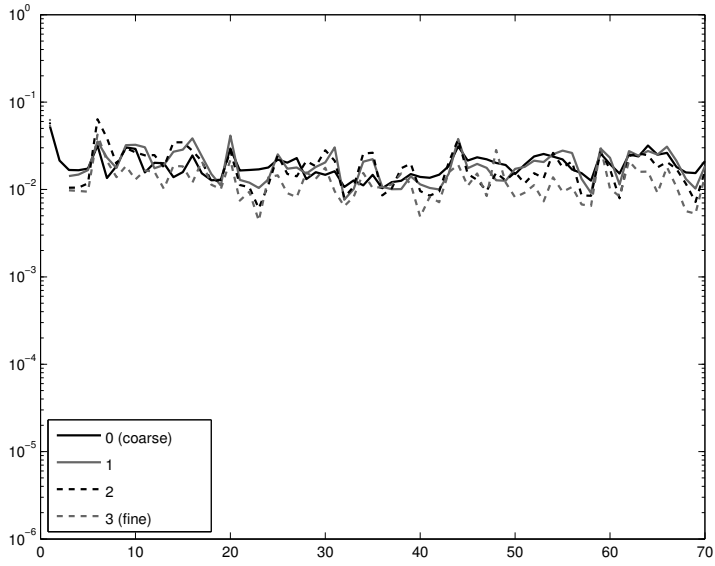
Backward error. — In order to verify their analysis of Section 6.2, Gratton and Toint (2010) compute the relative perturbations $\|E_i\|_\infty / \|\nabla^2 f(x_{k+1})\|_\infty$ during runs of the L-BFGS method using the Local and Coarse first strategies, and $\tau = 1.0$. Figure 6.10 shows typical results for the problems `P2D` and `MINS-SB`.

We observe that the relative size of the Hessian perturbation needed to make the approximate secant equations hold exactly is very modest (a few percent of $\|\nabla^2 f(x_{k+1})\|_\infty$, typically). Indeed, for non quadratic problems (see Figure 6.10b), the relative size of the Hessian perturbation necessary to make the exact secant equation (4.12) hold exactly is of the same order as that for the approximate secant equations in the early iterations of the algorithm, and ultimately decreases to the order of 10^{-5}. For quadratic functions, it is invisible on Figure 6.10a, since it is always tiny (between 10^{-15} and 10^{-10}).

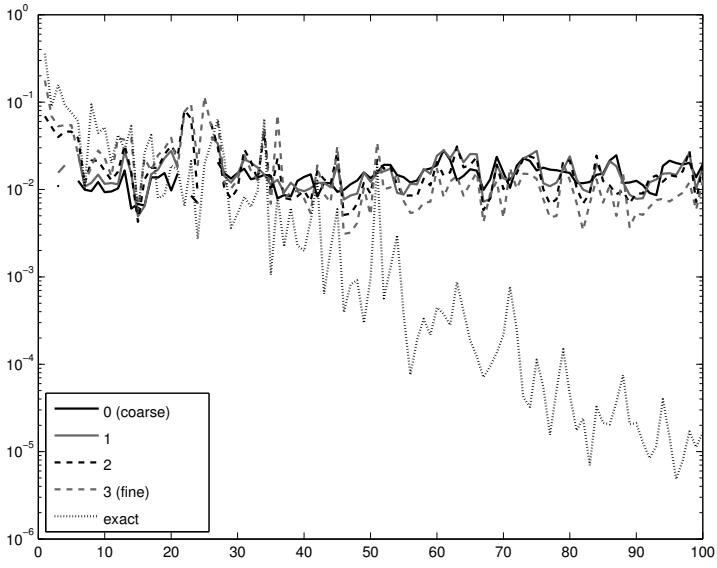
Robustness to secant pairs perturbation. — We hence wonder how these perturbations affects the convergence of the global process. In order to test the robustness of the L-BFGS method to these perturbations, we add at each iteration a random perturbation (of fixed size η) to the secant pairs. Every secant pair (s, y) is thus replaced by the pair

$$(s + \eta\|s\|u, y + \eta\|y\|v) \tag{6.12}$$

for some random unit vectors u and v. Figure 6.11 on the facing page displays the typical evolution of the number of iterations needed to achieve convergence (we required the norm of the gradient to be less than 10^{-5}) with the relative size η of the perturbation.

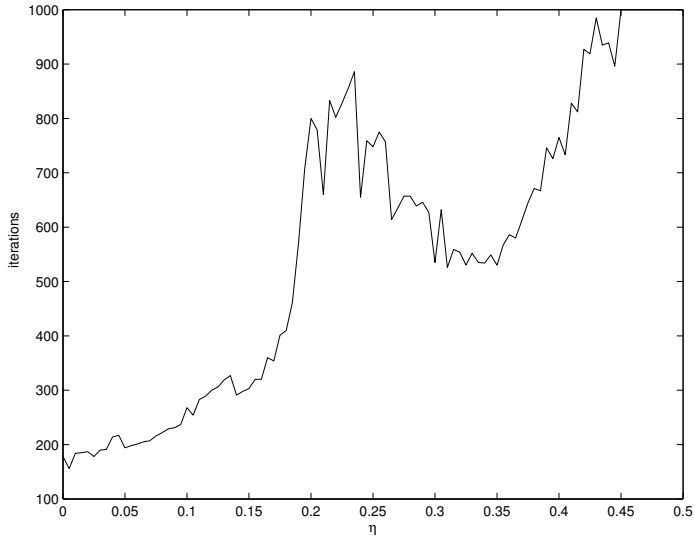


(a) *Problem* P2D, $n = 63^2$.

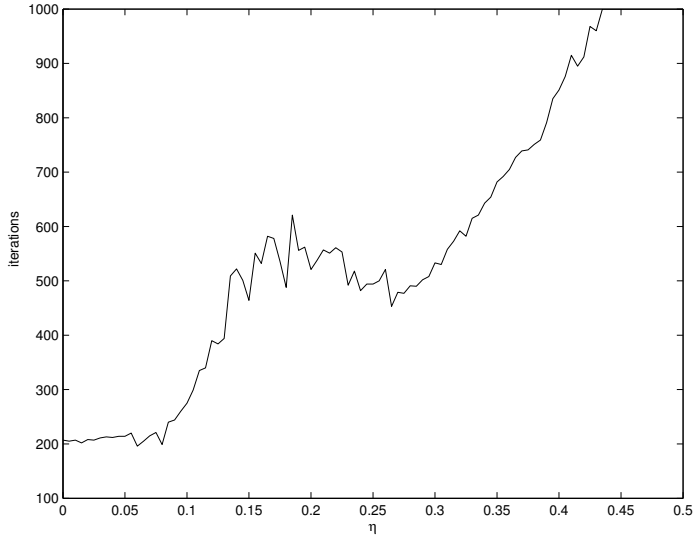


(b) *Problem* MINS-SB, $n = 63^2$.

Figure 6.10 — *Evolution of the relative Hessian perturbation sizes* ($\|E_i\|_\infty / \|\nabla f(x_{k+1})\|_\infty$) *with the iteration counter* k (*Local* and *Coarse first strategies,* $m = 7$, $r = 4$, *logarithmic vertical scale).*



(a) *Problem* `P2D`, $n = 63^2$.



(b) *Problem* `MINS-SB`, $n = 63^2$.

Figure 6.11 — *Evolution of the number of iterations needed for the L-BFGS method to converge, with the relative size of the perturbations imposed on the secant pairs.*

As expected, we observe an increase of the number of iterations needed to converge with the size of the perturbations, as more and more noisy and misleading information is brought to the L-BFGS algorithm. However, little effect appears when the size of the perturbations is less than a few percents.

6.5.2 Smoothed secant pairs novelty

In an attempt to understand why smoothed *approximate* secant pairs give better results than past *exact* pairs, we remember that the L-BFGS method and the conjugate gradient method are equivalent on quadratic problems. Restarting the conjugate gradient method and using then the L-BFGS preconditioner moreover does not change the iterate sequence. Only *new* information, in the sense that the secant direction is conjugate to the space $\mathcal{T}_k = \text{span}\{s_j \,:\, j \leqslant k\}$ spanned by the previous steps, is useful for that method.

We guess that the smoothed steps indeed bring at the current iteration such new information, which would otherwise have been only encountered in future iterations. Contrary to the past exact secant pairs, this should yield a more effective preconditioning of the conjugate gradient method. We test this claim by projecting the smoothed steps onto the $\nabla^2 f(x_k)$-conjugate complementary space of $\mathcal{T}_k$. Figure 6.12 thus represents the evolution of the relative norm of the projection of the smoothed steps $s_{k,i}$ onto

$$\mathcal{T}_k^{\mathsf{c}} = \left\{x \in \mathbb{R}^n : \left\langle x, \nabla^2 f(x_k) z \right\rangle = 0,\ \forall\, z \in \mathcal{T}_k\right\} \tag{6.13}$$

As expected, the smoothed steps turn out to really have a significant component in that complementary space $\mathcal{T}_k^{\mathsf{c}}$.

The concept of *new information* should also be examined with respect to the error $e_k = x_k - x_*$ on the solution. In the quadratic case, the projection of this error on $\mathcal{T}_{k-1}$ is known to be zero with the conjugate gradient method (see for instance Lemma 5.1.5 of Conn *et al.*, 2000). Figure 6.13 displays the projection of the error e_k on $\mathcal{T}_{k-1}$ for the quadratic problem `P2D` using an exact linesearch in the L-BFGS method, and either the L-BFGS, or the Local, or the Mless strategy. We observe that for the classical L-BFGS method, the projected error is near machine precision as expected (up to numerical errors). When using the smoothed secant pairs, the projected error still remains rather small (at most a few percents of the error).

Using an inexact linesearch in the L-BFGS method may obviously break these results as the step directions are no more perfectly conjugated, but we wonder to what extent. Figure 6.14 on page 135 shows the same quantity for the problems `P2D` and `MINS-SB`, but using this time the Dennis-Schnabel linesearch. Interestingly, we observe that the relative projected error is significant, but is in average smaller with the Local and Mless strategies than with L-BFGS one. Using the approximate secant pairs therefore seems to produce an error that lies with more accuracy in $\mathcal{T}_k^{\mathsf{c}}$. The fact that the smoothed secant pairs mostly

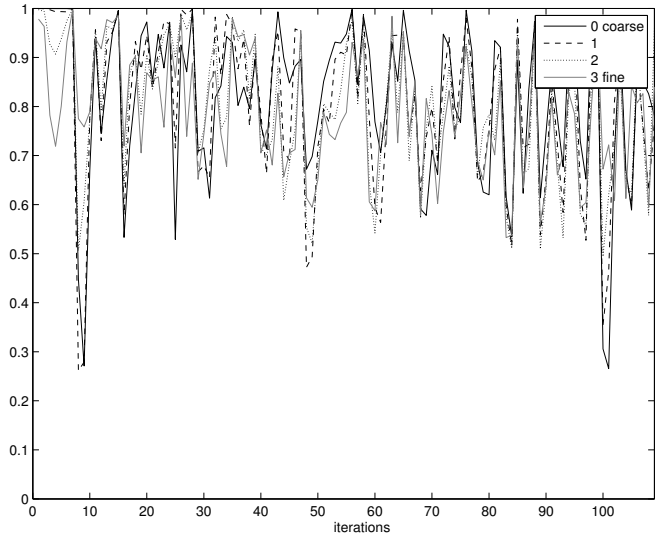


Figure 6.12 — *Evolution of the relative norm of the projection of the smoothed steps $S_i s_k$ onto $\mathcal{T}_k^c$ on problem* `MINS-SB` *(Local and Coarse first strategies, $n = 63^2$).*

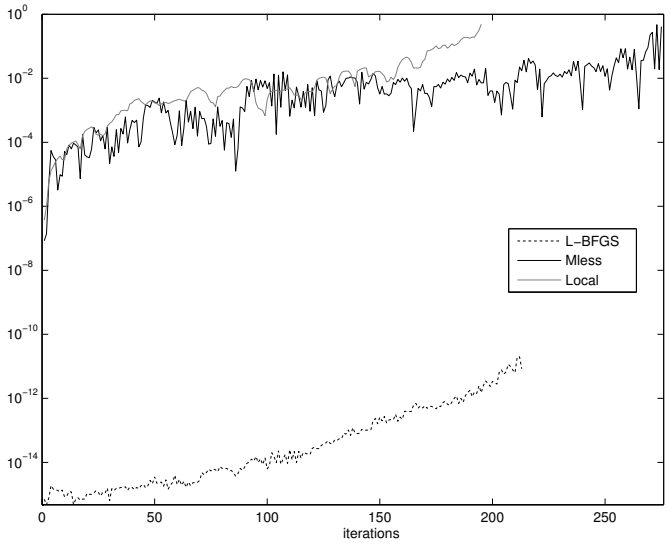
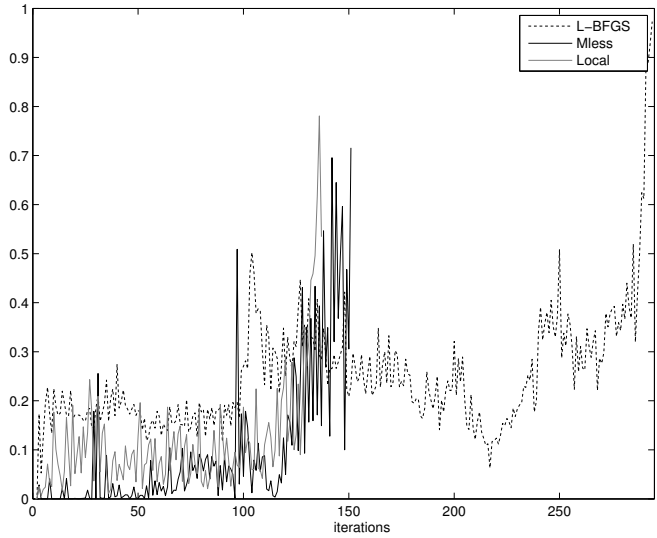
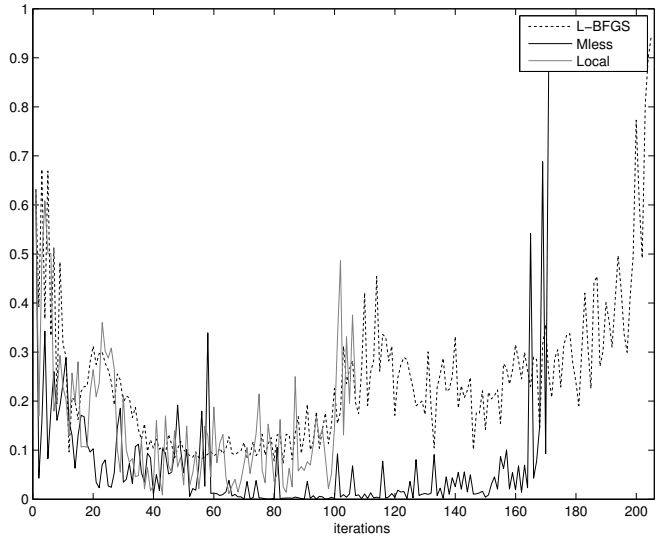


Figure 6.13 — *Relative projected error for problem* `P2D` *($n = 127^2$) with an exact linesearch (logarithmic vertical scale).*



(a) *Problem* P2D, $n = 127^2$.



(b) *Problem* MINS-SB, $n = 63^2$.

Figure 6.14 — *Relative projected error an inexact linesearch (Dennis-Schnabel).*

lie in that same space, should hence be an advantage to reduce the error more efficiently than with the exact pairs, which lie in $\mathcal{T}_k$.

6.6 Multiple secant equations satisfaction

In the previous experiments, we design the Hessian approximation plugging an ordered set of secant pairs into the L-BFGS formula. However, it is unclear how these pairs are tackled in that case. In particular, what is the accuracy achieved by the so-constructed Hessian approximation on every corresponding secant equation? We only know for sure that the secant equation for the last integrated pair is exactly satisfied. It is therefore the intent of Subsection 6.6.1 to examine more carefully the behaviour of the L-BFGS approximation, considering specifically this aspect. This obviously raises the question of designing a Hessian approximation in the context of multiple secant equations. This problem has already been considered by Schnabel (1983) (see also the discussion by Byrd, Nocedal and Schnabel, 1994). If we impose that every secant equation holds exactly, we may for instance compute the multiple secant version of the BFGS formula:

$$B_{k+1} = B_{k,0} - B_{k,0}S(S^T B_{k,0} S)^{-1} S^T B_{k,0} + Y(Y^T S)^{-1} Y^T, \tag{6.14}$$

where the columns of the matrices S and Y are the vectors $\{s_{k,j}\}_{j=1}^{m_k}$ and $\{y_{k,j}\}_{j=1}^{m_k}$, respectively. However, Schnabel (1983) shows that, if every secant equation holds exactly, the matrix $S^T Y$ needs to be symmetric and positive definite to ensure the symmetry and positive definiteness, respectively, of the approximated Hessian. Those are however assumptions that are tough to ensure in practice (see again Schnabel, 1983, for a method to alter the secant pairs such that these conditions hold).

The approach that we consider in Subsection 6.6.2 is the penalization of the secant equations violation. We hence define a new Hessian approximation that controls the relative violation of the secant equations. Secant pairs may thus serve the construction of the Hessian approximation according to how worthy we believe they are, depending on some criterion like their accuracy or novelty.

6.6.1 Behaviour of the L-BFGS method

In an attempt to see through the L-BFGS "black-box", we perform a few experiments considering some multilevel problems, namely `BRATU`, `DNT`, `MINS-SB` and `P2D`. We report only here the results for the problem `DNT` (with 127 variables) to facilitate the reading, but the trends for the other problems are quite similar. We also report in Appendix B the same graphs as in this subsection but for problem `BIGGS6`, which is not a multilevel one, for the sake of comparison. These experiments were performed using MATLAB®.

Secant equations violation. — We first measure the relative error

$$\varepsilon_{k,j} \stackrel{\text{def}}{=} \frac{\|y_{k,j} - B_{k+1} s_{k,j}\|}{\|\nabla^2 f(x)\| \|s_{k,j}\|} \tag{6.15}$$

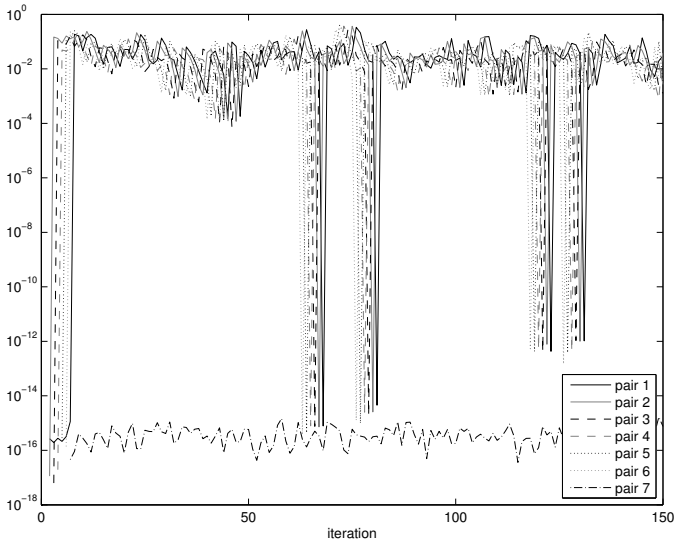
on the secant equations along the iterations generated by the classical L-BFGS algorithm. Figure 6.15 shows the evolution of this measure on problem DNT, fixing the memory limit m to 7 and using the Dennis-Schnabel linesearch (each curve corresponds to a fixed j). In Figure 6.15a, the points displaying $\varepsilon_{k,j}$ for a common k are vertically aligned. In Figure 6.15b, we instead align vertically the measures $\varepsilon_{k,j}$ that describes the life of a given pair (that generated at iteration k), from the index $(k, 7)$ to $(k + 6, 1)$.

First, we observe that the last introduced secant equation is fulfilled near machine precision as expected, and that the accuracy on the other secant equations is generally of a few percents. In contrast to our *a priori* opinion, the behaviour of the L-BFGS method with respect to each integrated secant pair seems to depend more on the pair intrinsic quality than on its numbering in the sets $\mathcal{P}_k$. Indeed, the curves are remarkably superposed in Figure 6.15b. We also observe that each time the linesearch requires more than one function evaluation (that is when the Wolfe conditions are not fulfilled by the quasi-Newton step $d_k = -H_k g_k$), the accuracy on the secant equation introduced at that iteration k is really better (producing a peak in the figures), and that this effect remains for that particular secant equation as long as it is used in the L-BFGS approximation. This means that the L-BFGS method somehow implicitly registered the directions in which troubles have occurred (that is when the quasi-Newton step was not satisfactory), and take more care to the information gathered in those directions. This tends to corroborate our theory on the need of new pertinent information. Indeed, if the quasi-Newton step did not fulfil the Wolfe condition, it means that the second-order information available at that moment was too approximate in comparison to the information brought by the new secant pair, which then constitutes (relatively) "new information".

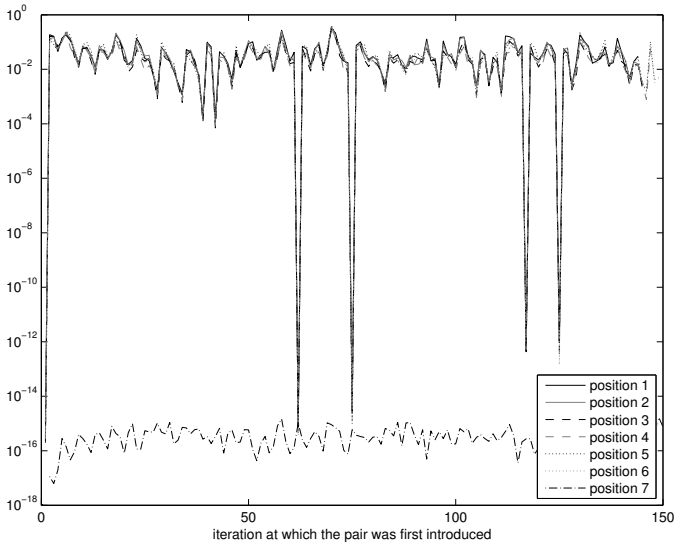
Conjugation of the secant directions. — Trying to explain these peaks in the graphs, we examine the conjugation of the smoothed steps $s_{k,j}$. More precisely, Figure 6.16a displays the relative norm of the orthogonal projection of each $s_{k,i}$ on the other $y_{k,j}$ $(j \neq i)$:

$$\frac{\| \mathrm{P}_{\mathrm{span}\{y_{k,j} \,:\, j \neq i\}} \, s_{k,i} \|}{\|s_{k,i}\|}.$$

where $\mathrm{P}_{\mathrm{span}\{y_{k,j} : j \neq i\}}$ is the orthogonal projection on the space spanned by the vectors $y_{k,j}$ for $j \neq i$. We observe that only the first secant direction reaches conjugation peaks, which correspond to the peaks visible on Figure 6.15 (describing the accuracy on the secant equations). This seems to indicate that the



(a) *Pairs used at the same iteration are vertically aligned.*



(b) *All uses of the same pair are vertically aligned.*

Figure 6.15 — *Relative error $\varepsilon_{k,j}$ for the secant pairs generated by L-BFGS on problem* `DNT` *($m = 7$, Dennis-Schnabel linesearch).*

better conjugation of the direction is some consequence of the better accuracy on the secant equations. We test this claim by considering the projection only on the pairs generated after the tested one:

$$\frac{\| \mathrm{P}_{\mathrm{span}\{y_{k,j} \,:\, j>i\}} \, s_{k,i} \|}{\| s_{k,i} \|}.$$

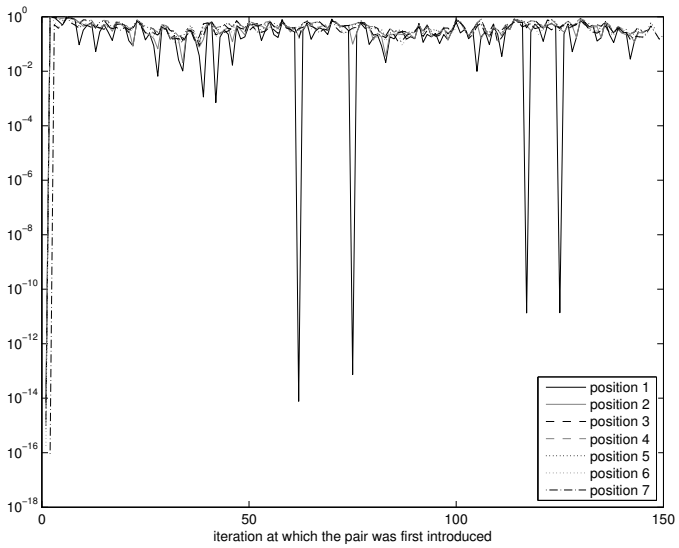
The results of this experiment are displayed in Figure 6.16b. We observe again the overlapping phenomenon, and the peaks of conjugation are this time synchronized with the peaks of accuracy on the secant equations for every curve.

Influence of the linesearch. — As we observe an influence of the linesearch process, we perform another series of test with the linesearch of Moré and Thuente (1994), which uses the strong Wolfe condition, with the same classical value of the parameters $\delta = 10^{-4}$ and $\sigma = 0.9$. Figure 6.17 on page 141 displays the same quantity $\varepsilon_{k,j}$ as before, but using the Moré-Thuente linesearch. In this figure, the overlapping phenomenon is not so visible, perhaps because the curves are not so oscillatory. Moreover, the effect of the pairs order is visible at the first iterations, but is mild. In all these experiments, the quasi-Newton step fulfils the strong Wolfe condition (except at the very first iterations), and significant accuracy improvements are no more observed.

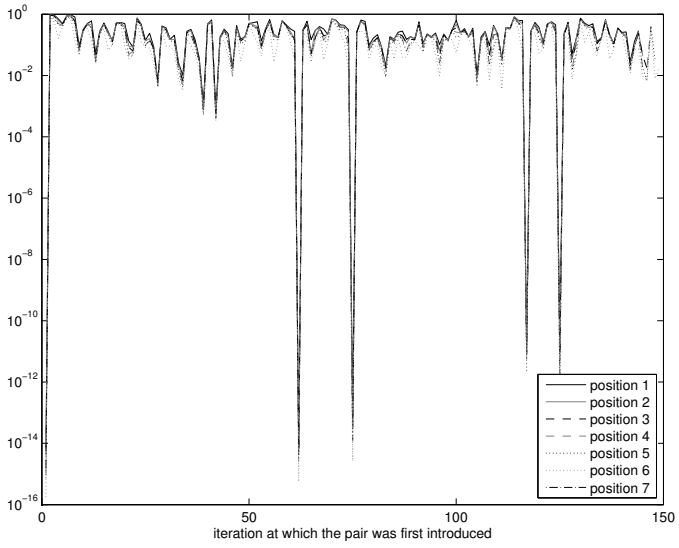
We fixed σ to 0.7 to strengthen the strong Wolfe condition and attempt to refuse the simple quasi-Newton step. The corresponding experiment is displayed in Figure 6.18 on page 142. This time, we observe the same kind of behaviour as with the Dennis-Schnabel line search: the curves are overlapping, and there are some peaks of accuracy. However, no more correlation can be found for these peaks with the number of function evaluations required by the linesearch (which is generally two for this value of σ).

Influence of the memory size. — We consider the effect of using very few/ many secant pairs in the L-BFGS update (3, 7, 9, 15, 50 and 150). Figure 6.19 on page 143 displays the results of this experiment with 3 and 50 pairs allowed in memory. Our observations remain the same if more or less of secant pairs are kept in memory to define the Hessian approximation. The overlapping phenomenon is remarkably visible for so many curves. Notice also that the L-BFGS method turns back to the BFGS method in the case $m = 150$; we then observe that the peaks behaviour on the graphs seems not to be caused by the limited memory, but to be already present in the classical BFGS method.

Keeping "new" information in memory. — As the L-BFGS method pays more attention to some secant pairs, we imagine a variant that keeps in memory the secant pairs for which the Dennis-Schnabel linesearch takes more than one (inner) iteration. These particular pairs are accumulated in a separate set of size at most $m - 1$, and are used first in the L-BFGS update. The preliminary

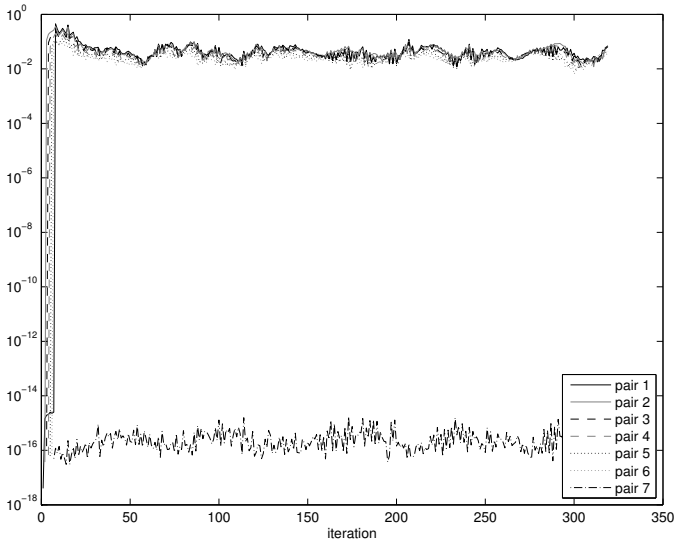


(a) *Conjugacy to the other directions* $s_{k,j}$ $(j \neq i)$.

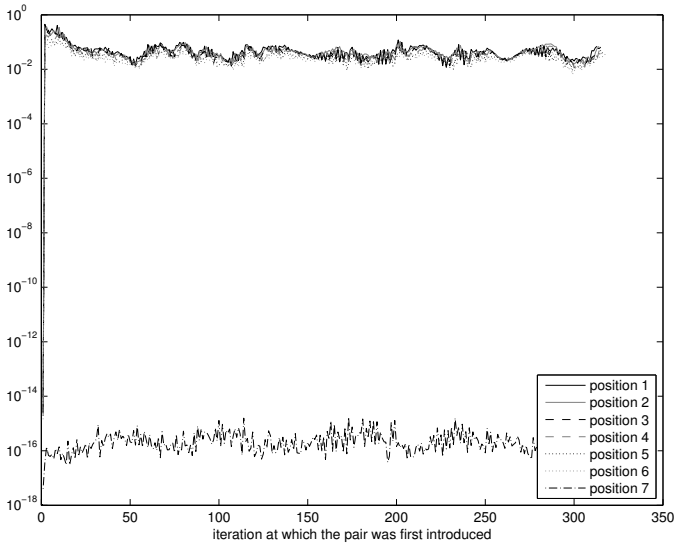


(b) *Conjugacy to the newer directions* $s_{k,j}$ $(j > i)$.

Figure 6.16 — *Conjugation of direction* $s_{k,i}$ *with respect to other directions in* $\mathcal{P}_k$*, for the problem* `DNT` *(*$m = 7$*, Dennis-Schnabel line-search). All uses of the same pair are vertically aligned.*

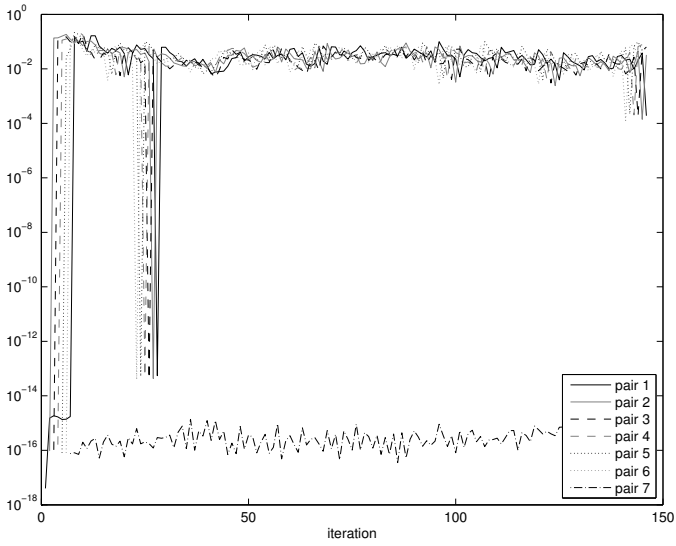


(a) *Pairs used at the same iteration are vertically aligned.*

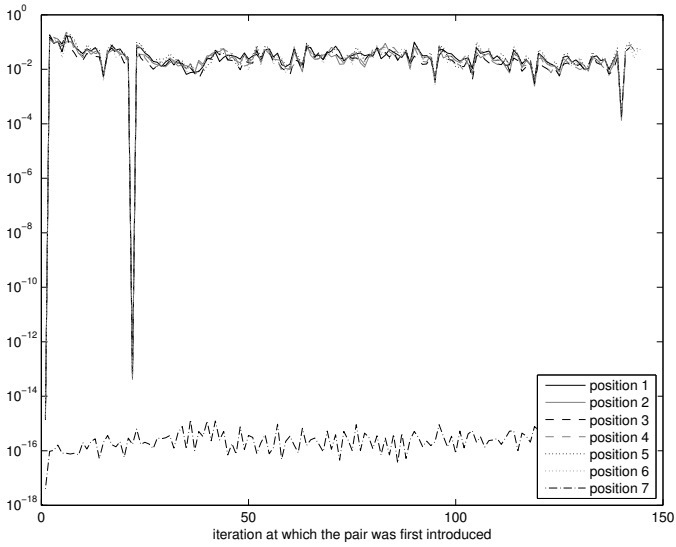


(b) *All uses of the same pair are vertically aligned.*

Figure 6.17 — *Relative error $\varepsilon_{k,j}$ for problem* `DNT` *($m = 7$, Moré-Thuente linesearch with $\sigma = 0.9$).*

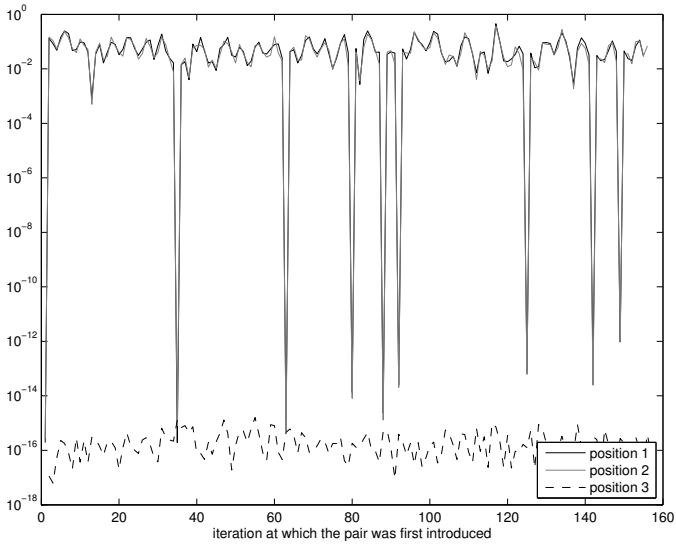


(a) *Pairs used at the same iteration are vertically aligned.*

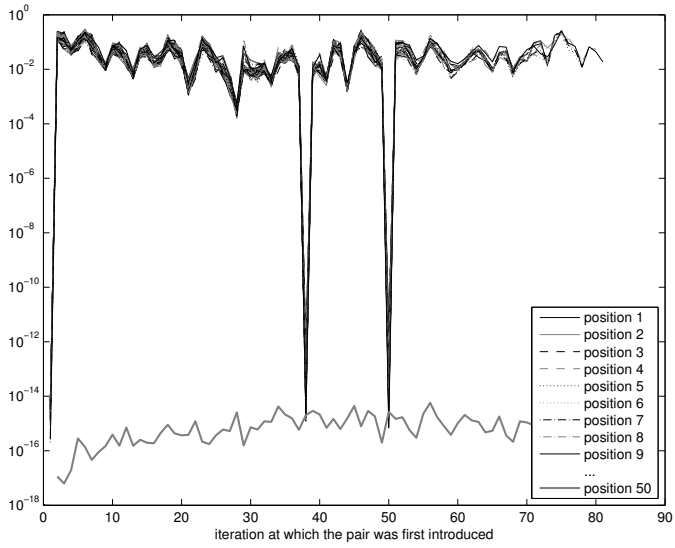


(b) *All uses of the same pair are vertically aligned.*

Figure 6.18 — *Relative error $\varepsilon_{k,j}$ for problem* `DNT` *($m = 7$, Moré-Thuente linesearch with $\sigma = 0.7$).*



(a) $m = 3$.



(b) $m = 50$.

Figure 6.19 — *Relative error $\varepsilon(s_i, y_i)$ for problem* `DNT` *(Dennis-Schnabel linesearch). All uses of the same pair are vertically aligned.*

results presented in Table 6.4, show no evidence of a better convergence speed with this new secant pair selection strategy.

Problem	n	m	$\tilde{m}$	f	$\|g\|$	#f	#g	iter
DNT	127	7	–	$1.39 \cdot 10^{-10}$	$8.94 \cdot 10^{-06}$	140	136	135
P2D	961	7	–	$-2.15 \cdot 10^{+02}$	$7.69 \cdot 10^{-06}$	97	93	92
MINS-SB	961	7	–	$1.09 \cdot 10^{+00}$	$9.96 \cdot 10^{-06}$	109	107	105
BRATU	961	5	–	$-3.10 \cdot 10^{-01}$	$8.90 \cdot 10^{-06}$	95	91	90

(a) *L-BFGS.*

Problem	n	m	$\tilde{m}$	f	$\|g\|$	#f	#g	iter
DNT	127	7	4	$1.81 \cdot 10^{-10}$	$9.90 \cdot 10^{-06}$	157	153	152
P2D	961	7	2	$-2.15 \cdot 10^{+02}$	$9.53 \cdot 10^{-06}$	84	82	81
MINS-SB	961	7	4	$1.09 \cdot 10^{+00}$	$9.63 \cdot 10^{-06}$	126	123	121
BRATU	961	5	3	$-3.10 \cdot 10^{-01}$	$6.86 \cdot 10^{-06}$	88	85	84

(b) *L-BFGS with "keeping new pairs" strategy.*

Table 6.4 — *Results for the classical L-BFGS method and its variant with the "keeping new pairs" strategy, on some multilevel problems ($\tilde{m}$ is the number of pairs eventually stored). The columns f, g,* #f, #g *and* iter *hold the final function value, final gradient norm, and the numbers of functions, gradients and iterations needed to converge, respectively.*

Conclusion. — In this subsection, we observe that the L-BFGS approximation satisfies the secant equations to a few percents, except for the last one that it integrates. On our set of multilevel problems, the intrinsic quality of the secant pairs appears more determinant of the accuracy on the respective secant equations than its position in the L-BFGS update (except for the last one obviously). In particular, the method takes more care to the secant pairs produced at iterations for which the currently known information was too approximate for the quasi-Newton to be satisfactory, also inducing a better conjugacy of the steps produced afterwards with respect to that privileged ones. We observe that this behaviour is independent of the memory limit, and is in fact already present with the classical BFGS update. The problem of finding an appropriate justification to the L-BFGS method behaviour remains open, though.

6.6.2 Penalization of the secant equations violation

After having observed how the L-BFGS process tackles multiple secant pairs, we intend to design a new Hessian updating procedure that also considers multiple secant pairs, but with more control and flexibility. To avoid clumsy notations,

we assume that m pairs of the form (s_i, y_i) $(i = 1, \ldots, m)$ are at our disposal. No reference to the iteration counter is therefore made in this subsection. The secant pairs need not come from iteration steps or their smoothed versions, but can be quite general. We denote the current Hessian approximation by B and the new approximation to design by B^+. The matrix E holds the correction $B^+ - B$. The vectors s_i and y_i form the columns of the $n \times m$ matrices S and Y, respectively.

As we mentioned in Chapter 4, most secant updates may be derived through a variational approach. Therefore, to define multisecant versions of the classical formulae, we should solve the constrained variational problem

$$\min_E \quad \frac{1}{2}\|W^{-T}EW^{-1}\|_F^2 \tag{6.16a}$$

$$\text{s.t.} \quad (B+E)s_i = y_i \text{ for } i = 1, \ldots, m, \quad \text{and } E = E^T, \tag{6.16b}$$

taking some particular nonsingular matrix W, whose choice determines the formula that is obtained. For instance, taking $W = I$ yields the multisecant PSB update, while any matrix W such that $W^T W S = Y$ yields the multisecant DFP update. The multisecant BFGS may be obtained by duality from its DFP counterpart.

Instead of solving (6.16), we aim to solve the penalized problem

$$\min_E \quad \frac{1}{2}\|W^{-T}EW^{-1}\|_F^2 + \frac{1}{2}\sum_{i=1}^m \omega_i \|(B+E)s_i - y_i\|_{\hat{W}^{-1}}^2 \tag{6.17a}$$

$$\text{s.t.} \quad E = E^T, \tag{6.17b}$$

where $\hat{W} = W^T W$, so relaxing the secant equations with nonnegative penalty parameters ω_i (for $i = 1, \ldots, m$).

Solving the penalized variational problem. — Zeroing the derivative of the Lagrangian function of this problem with respect to E in (matrix) direction K, we get

$$\operatorname{tr}(W^{-1}EW^{-T}W^{-T}KW^{-1}) + \operatorname{tr} K^T(M - M^T) + \sum_{i=1}^m \omega_i s_i^T K^T \hat{W}^{-1}((B+E)s_i - y_i) = 0, \tag{6.18}$$

where M is the Lagrange multiplier corresponding to the symmetry constraint. Using the vectorization operator vec, this equation rewrites as

$$\left\langle \operatorname{vec}(\hat{W}^{-1}E\hat{W}^{-1} + M - M^T), \operatorname{vec} K \right\rangle + \sum_{i=1}^m \omega_i \left\langle s_i \otimes \hat{W}^{-1}Es_i - s_i \otimes \hat{W}^{-1}r_i, \operatorname{vec} K \right\rangle = 0, \tag{6.19}$$

where $r_i = y_i - Bs_i$ is the residual on the i-th secant equation at the current Hessian. As the equation (6.19) must hold for every matrix K, we thus have

$$\begin{aligned} &\text{vec}(\hat{W}^{-1}E\hat{W}^{-1} + M - M^T) + \sum_{i=1}^{m} \omega_i (s_i \otimes \hat{W}^{-1}Es_i - s_i \otimes \hat{W}^{-1}r_i) = 0, \\ \Leftrightarrow \qquad &\hat{W}^{-1}E\hat{W}^{-1} + M - M^T + \sum_{i=1}^{m} \omega_i (\hat{W}^{-1}Es_i s_i^T - \hat{W}^{-1}r_i s_i^T) = 0. \end{aligned}$$

By summing this equation with its transpose and afterwards, pre- and post-multiplying by $\hat{W}$, we eliminate the Lagrangian multiplier M, and obtain that

$$E\left(I + \sum_{i=1}^{m} \omega_i s_i s_i^T \hat{W}\right) + \left(I + \sum_{i=1}^{m} \omega_i \hat{W} s_i s_i^T\right) E = \sum_{i=1}^{m} \omega_i (\hat{W} s_i r_i^T + r_i s_i^T \hat{W}).$$

We define the $n \times m$ matrices $R = Y - BS$, $\bar{R} = R\Omega$, $\bar{S} = S\Omega$, and $\bar{Z} = \hat{W}\bar{S}$, where $\Omega = \text{diag}(\sqrt{\omega}_i)$. Hence, we have to solve the Lyapunov equation

$$AE + EA^T = C, \tag{6.20}$$

where $A = I + \bar{Z}\bar{S}^T$ and $C = \bar{R}\bar{Z}^T + \bar{Z}\bar{R}^T$.

We claim that the solution E of that equation lies in the range of the matrix $[\bar{Z} \,|\, \bar{R}]$, which has an orthonormal basis Q. Indeed, let

$$E = \begin{pmatrix} Q & P \end{pmatrix} \begin{pmatrix} X & X_P^T \\ X_P & X_{PP} \end{pmatrix} \begin{pmatrix} Q^T \\ P^T \end{pmatrix}, \tag{6.21}$$

with X and X_{PP} are symmetric and where $[Q \,|\, P]$ is an orthogonal matrix. We know that $P^T\bar{R} = P^T\bar{Z} = 0$ and thus, $P^TC = 0$ and $A^TP = P$. Using (6.20), and then substituting E in $P^T(AE + EA^T)P = P^TCP$ hence gives

$$\begin{aligned} P^TA \begin{pmatrix} Q & P \end{pmatrix} \begin{pmatrix} X & X_P^T \\ X_P & X_{PP} \end{pmatrix} \begin{pmatrix} Q^T \\ P^T \end{pmatrix} P \\ + P^T \begin{pmatrix} Q & P \end{pmatrix} \begin{pmatrix} X & X_P^T \\ X_P & X_{PP} \end{pmatrix} \begin{pmatrix} Q^T \\ P^T \end{pmatrix} A^TP = 0, \end{aligned}$$

and finally, $X_{PP} = 0$. Then, using $(AE + EA^T)P = CP$, we also have

$$(A + I_n)QX_P^T = 0,$$

and thus $X_P = 0$ provided that $2I_n + \bar{Z}\bar{S}^T$ is non-singular. This leads to the reformulation

$$(Q^TAQ)X + X(Q^TAQ)^T = Q^TCQ, \tag{6.22}$$

which is a new Lyapunov equation, but with a smaller $2m \times 2m$ unknown matrix X to determine. We solve equation (6.22) using the method of Bartels and

Stewart (1972), which first computes the real Schur decomposition of Q^TAQ: $N = U^T(Q^TAQ)U$ (U orthogonal and N quasi-upper triangular) and then back-substitutes for Ξ in

$$N\Xi + \Xi N^T = U^TQ^TCQU.$$

Every column of Ξ may indeed be determined from the last to the first one. The solution of (6.20) is therefore given by $E = QU\Xi U^TQ^T$.

Numerical cost. — This procedure requires $4m^2n$ flops to compute the matrix Q (by a thin QR decomposition for instance), $2m^2n + O(m^3)$ flops to form Q^TAQ, and $O(m^3)$ flops to solve (6.22), which is marginal in the context of limited memory methods applied on large-scale problems. It takes $2mn^2 + 12m^2n$ additional flops to form E, but only $4mn + 12m^2$ to compute the product of E by a vector.

6.6.3 Some penalized Hessian approximations

We now present some practical updates that may be obtained with the framework developed in the previous subsection, letting the computational details to Appendix C.

Penalized PSB update. — If we consider a single secant pair (s, y) and take $W = I$, the solution of (6.17) gives the penalized PSB update:

$$B^+ = B + \frac{rs^T + sr^T}{2\omega^{-1} + \|s\|^2} + \left(\frac{1}{\omega^{-1} + \|s\|^2} - \frac{2}{2\omega^{-1} + \|s\|^2}\right)\frac{\langle s, r\rangle}{\|s\|^2}ss^T, \quad (6.23)$$

where $r = y - Bs$. Observe that letting ω go to infinity gives back the PSB correction scheme (4.17) on page 85.

Penalized DFP update. — If we still consider a single secant pair (s, y), but now choose a matrix W such that $W^TWS = Y$, solving (6.17) yields the penalized DFP update:

$$B^+ = B + \frac{ry^T + yr^T}{2\omega^{-1} + \langle s, y\rangle} + \left(\frac{1}{\omega^{-1} + \langle s, y\rangle} - \frac{2}{2\omega^{-1} + \langle s, y\rangle}\right)\frac{\langle s, r\rangle}{\langle s, y\rangle}yy^T. \quad (6.24)$$

Observe again that letting ω go to infinity gives back the DFP correction scheme (4.20) on page 86.

Penalized BFGS update. — The penalized inverse BFGS formula is obtained by duality from its direct DFP counterpart, since it is the solution of the same

variational problem but with $S \leftrightarrow Y$ and $W \leftrightarrow W^{-1}$. In case of a single secant pair (s, y), we thus obtain

$$H^+ = H + \frac{ps^T + sp^T}{2\omega^{-1} + \langle s, y\rangle} + \left(\frac{1}{\omega^{-1} + \langle s, y\rangle} - \frac{2}{2\omega^{-1} + \langle s, y\rangle}\right)\frac{\langle p, y\rangle}{\langle s, y\rangle} ss^T \quad (6.25)$$

$$= \left(I - \frac{\rho s y^T}{1 + 2\rho\omega^{-1}}\right) H \left(I - \frac{\rho y s^T}{1 + 2\rho\omega^{-1}}\right) + \theta\rho ss^T, \quad (6.26)$$

with $\rho = \langle s, y\rangle^{-1}$, $p = s - Hy$ and

$$\theta = \left(1 + \tfrac{\rho}{\omega}\right)^{-1} - \rho\langle y, Hy\rangle\left[\left(1 + \tfrac{2\rho}{\omega}\right)^{-2} + \left(1 + \tfrac{\rho}{\omega}\right)^{-1} - \left(\tfrac{1}{2} + \tfrac{\rho}{\omega}\right)^{-1}\right].$$

6.7 Perspectives

The approximate secant pairs generated by smoothing the exact secant pair on the multilevel hierarchy have been proved highly valuable. But the order in which they are used in the L-BFGS recursion remains an open subject of research. Our numerical experiments did not manage to indicate an ordering particularly better than the others. From our quick analysis of the L-BFGS behaviour, we may even think that this ordering is not so important, but that the secant pairs quality prevails. The matter is therefore to pick out particularly interesting pairs, and to preferably use them.

A deeper study of the secant pairs quality could lead to appropriate penalization parameters for our new multisecant updating procedure. In our preliminary numerical tests of this algorithm, we indeed met no better results than with the classical L-BFGS formula, but one still ignores whether this is due to the new updating itself or to our poor guess of penalization parameters. Further research are therefore needed to decide the practical interest of this new Hessian approximation method.

Part III

Application and software

Chapter 7

An application to snake-skin pigmentation patterns

This chapter presents an application of the RMTR method combined with the LTS method for Hessian approximation to a topic in mathematical biology: the modelling of snake-skin pigmentation patterns.

Ever since the seminal theoretical developments by Turing (1952), numerous mathematical developments and practical applications have emerged about the solutions of nonlinear reaction-diffusion equations. More particularly, such systems have been widely investigated in the context of mathematical biology, notably to provide a deterministic modelling of the large variety of patterns observed in animals. Among those works, it is probably that of Murray (2003) and his collaborators that has most biological relevance for real patterns in animals. Their models are based on a simplistic interaction scheme between chromatophores and one additional chemical species, without taking into account the complexity of the underlying biochemical mechanisms. However, such simple models have been proved very efficient to capture several key features of the complexity in pattern generation, like stability and time-scale of relaxation (Baker, Schnell and Maini, 2009), bifurcation between different pattern topologies (Winters, Myerscough, Maini and Murray, 1990, and Maini, Myerscough, Winter and Murray, 1991), criticality in the parameters for the emergence of pattern organization (Chang, Wu, Baker, Maini, Alibardi and Chuong, 2009). We also refer to Barrio, Varea, Aragón and Maini (1999), Painter, Maini and Othmer (1999), Plaza, Sánchez-Garduño, Padilla, Barrio and Maini (2004), Lin, Jiang, Baker, Maini, Widelitz and Chuong (2009), Zhu, Zhang, Newman and Alber (2009) for other recent developments in this field.

The numerical solution of such nonlinear coupled reaction-diffusion systems is not a trivial task, even though the model equations, domain geometry and boundary conditions are usually strongly simplified compared to the reality

of biological systems. To accurately model such real situations, (many) more degrees of freedom have to be taken into account since pattern generation is a consequence of several overlapping bio-physico-chemical mechanisms, which are at play on the sophisticated geometry of real animals skins, a geometry that is very far from the idealized domain used in numerical simulations. Sophisticated algorithms are therefore to be used in the numerical solution of reaction-diffusion systems to model real biological systems. Those algorithms should be easily extensible to handle more coupled degrees of freedom, to incorporate new mechanisms in the model and to deal with complex domain geometries without becoming completely intractable from the numerical point of view.

This chapter reports our inceptive investigation in the numerical determination and analysis of patterns arising from these reaction-diffusion systems. More specifically, we consider here the determination of stationary patterns from the reaction-diffusion model of Maini *et al.* (1991) using multilevel optimization (in fact the RMTR method presented in Section 3.2). Before going to the heart of the matter, we start with a brief discussion in Section 7.1 on the solution of physical problems (possibly) modelled by PDEs. In Section 7.2, we then present the pigmentation model, several geometries on which we consider it, and the discretization scheme used to convert these equations in a tractable optimization problem. Section 7.3 is then devoted to numerical aspects of its resolution, and to the presentation of the obtained patterns.

In addition to developing powerful numerical methods for pattern generation, another challenge lies in characterizing the complexity in their organization through quantitative estimators. Indeed, comparing patterns, real ones or numerical solutions, is often made qualitatively (see for instance Maini *et al.*, 1991). Therefore, the questions of estimating model adequacy to real data, measuring pattern organization or studying nonlinear features like bifurcations or phase transitions are difficult to settle in absence of quantitative estimators adapted to pattern recognition. This is a necessary complementary work to numerical simulations as it will allow us to explain further the emergence of complexity in biological systems. This is why we propose in Section 7.4 to investigate two different families of methods that are of great importance in the study of complex systems. The first is based on the properties of Fourier transform of patterns, while the second exploits random walks on the discrete graph of the discretization cells that constitutes a numerical pattern. These methods will be shown to be rather complementary since the first treats the properties of periodic constitutive elementary patterns, while the second gives information on sharpness and global organization of the pattern.

We finally conclude this chapter (Section 7.5) with some perspectives on the potential future work, notably by considering a finite-element discretization.

7.1 Introduction to the solution of PDEs

Many physical problems may be formulated as finding stationary points of some functional, called *action*. Note that, in many cases, these points are in fact known to be minimizers of the action. Consider now an action of the form

$$J(y) = \int_{\mathcal{D}} L(x, y, y_{x_1}, \ldots, y_{x_n})\, dx, \tag{7.1}$$

where $y : \mathcal{D} \to \mathbb{R}$ is the unknown function, defined on the subset $\mathcal{D}$ of $\mathbb{R}^n$, and $y_{x_i} = \partial y / \partial x_i$. Each stationary point of this action J is then necessarily a solution of the so-called *Euler-Lagrange equation*

$$\frac{\partial L}{\partial y} - \sum_{i=1}^{n} \frac{\partial}{\partial x_i} \left(\frac{\partial L}{\partial y_{x_i}} \right) = 0 \tag{7.2}$$

(see Chapter 2 of Lánczos (1986) or Chapter 17 of Arfken (1970) for a more general definition).

The solution of the physical problem under consideration may therefore be considered using any of these formulations, either by computing a stationary point of the action or by solving the corresponding Euler-Lagrange partial differential equation. In both cases, the problem is typically discretized to be solved. In the variational approach, the integral is then approximated with numerical quadrature. By contrast, the PDE (7.2) may be discretized using either explicit or implicit methods (see Strikwerda, 2004). In the first case, the solution is computed by propagating some initial condition (initial value problem), whereas the second case yields a system of equations. If these equations are linear, then multigrid methods may be applied, and are especially efficient for elliptic problems (see Sections 1.1 and 4.7 of Trottenberg *et al.*, 2001). Otherwise, one may directly tackle the system of equations with Newton's method, or minimize the (squared) norm of the residual on these equations (see Chapter 11 of Nocedal and Wright (2006), on the solution of nonlinear equations).

If the (Euler-Lagrange) PDE is not elliptic, minimizing the corresponding action may be an ill-posed problem (the functional J may be unbounded below). Yet this situation makes sense in reality, since stationary points, and not only minimizers, should be sought. By contrast, minimizing the residual norm is always a well-posed optimization problem, though only global minimizers of the residual norm are solutions of the corresponding system of equations. If the residual norm is a convex function, then finding such a global minimizer boils down to finding a local minimizer, but may be tough or practically impossible otherwise. Moreover, the Hessian of the residual norm is typically denser and has a larger condition number than that of the discretized action, potentially increasing the iteration cost of optimization procedures in the residual approach (for instance, in the 2-D Laplace's problem, the Hessian of the residual norm is biharmonic, displaying 13 nonzero diagonals instead of 5 for the action). Note

also that applying Newton's method directly to the system of equations may fail if the starting point is not close enough from the solution.

To solve a boundary value problem, (explicit) propagation techniques can not be applied. We therefore have to consider the other methods. If the underlying PDE is linear, then any approach is suitable. However, if it is nonlinear, the variational approach is probably more efficient and should be considered first provided that the action is known and bounded below. Otherwise (and in particular for hyperbolic boundary value problems), a global minimizer of the residual norm must be computed, though one is reduced to hope that the generated local minimizer is also global if the residual norm is nonconvex. This process might still be helped (but without guarantee to obtain a global minimizer) using for instance non-monotone optimization algorithms, in which the function value is only required to decrease in average along the iterations (see for instance Grippo, Lampariello and Lucidi (1986, 1989, 1991), Zhang and Hager (2004), in the context of linesearch methods, and the discussion in Section 10.1 of Conn *et al.* (2000) for trust-region methods).

7.2 Modelling

We first present briefly in Subsection 7.2.1 the biological model of pigmentation patterns on snake skins that we consider in this application, namely the cell-chemotaxis model. Subsection 7.2.2 is then devoted to simplified geometries representing the snake skin and on which the model equations are posed. In particular, we describe how the classical operators ∇ and Δ of differential geometry are written on these surfaces given particular set of coordinates. This yields the model finite-difference discretization that we finally exhibit in Subsection 7.2.3.

7.2.1 Biological modelling

Generic reaction-diffusion systems are coupled parabolic systems of partial differential equations with nonlinear source terms:

$$\begin{aligned}\frac{\partial \mathbf{u}}{\partial t} &= D_u \Delta \mathbf{u} + f(\mathbf{u}, \mathbf{v}) \\ \frac{\partial \mathbf{v}}{\partial t} &= D_v \Delta \mathbf{v} + g(\mathbf{u}, \mathbf{v}).\end{aligned}$$

This system describes the evolution of concentrations of two chemicals $\mathbf{u}$ and $\mathbf{v}$ under the influence of diffusion (with constant diffusion coefficients D_u and D_v) and reaction (the source terms f and g). Such systems describe the diffusion-driven instability mechanism discovered by Turing (1952) that produces spatial patterns from almost homogeneous initial distributions.

Here, we focus on a similar mechanism of spatial pattern formation through chemotactic motion. In this mechanism, cells move via random diffusion preferentially in the direction of high gradients in chemical concentration (chemotaxis) while these gradients are amplified through local secretion by the cells. In the same time, the cells undergo proliferation, and the chemical diffuses randomly after being secreted by the cells. This mechanism, analogous to reaction-diffusion systems, is dubbed *cell-chemotaxis models*. More precisely, the interactions between cells and the chemical during their motion through space can be modelled in terms of the following PDE system (Oster and Murray, 1989):

$$\frac{\partial \mathbf{n}}{\partial t} = D\Delta \mathbf{n} - \alpha \nabla \cdot (\mathbf{n} \nabla \mathbf{c}) + r\mathbf{n}(N - \mathbf{n}) \tag{7.3a}$$

$$\frac{\partial \mathbf{c}}{\partial t} = \Delta \mathbf{c} + \frac{\mathbf{n}}{1 + \mathbf{n}} - \mathbf{c}. \tag{7.3b}$$

In the above system, the cell number density $\mathbf{n}$ and the activator chemical concentration $\mathbf{c}$ are the two fields to be determined. The parameters D, α, r and N are all positive-valued and explained further. The first term in the right-hand sides of (7.3) is a diffusive term accounting on large scales for the microscopic random motion of cells $\mathbf{n}$ and chemical $\mathbf{c}$ ($D \stackrel{\text{def}}{=} D_n/D_c$ is the ratio between their diffusion coefficients). The second term in (7.3a) models chemotaxis (with chemotactic parameter α). Finally, the last source terms of (7.3a) and (7.3b) represent cell proliferation through logistic growth term (with N being the carrying capacity) and chemical production with saturation for high cell density and linear decay, respectively. The nonlinearity in the derivatives brought by the chemotaxis term is a unique feature to be handled by numerical methods.

Equations (7.3) are subject to boundary conditions. Typically, one assumes that there is no flow of cells and chemicals through the boundaries of the domain $\mathcal{D}$, so we adopt zero-flux boundary conditions of the form

$$\langle u, \nabla \mathbf{n} \rangle = \langle u, \nabla \mathbf{c} \rangle = 0, \tag{7.4}$$

where u is a unit normal vector to the boundary $\partial\mathcal{D}$. In what follows, we will also adopt periodic boundary conditions depending on the domain geometry.

As mentioned in the introduction, we are first interested in the organization of stationary patterns, leaving the question of the time-evolution of this organization for further studies. We therefore look for time-independent solutions to (7.3), namely by solving the following set of coupled PDEs:

$$D\Delta \mathbf{n} - \alpha \nabla \cdot (\mathbf{n} \nabla \mathbf{c}) + r\mathbf{n}(N - \mathbf{n}) = 0, \tag{7.5a}$$

$$\Delta \mathbf{c} + \frac{\mathbf{n}}{1 + \mathbf{n}} - \mathbf{c} = 0. \tag{7.5b}$$

Observe that the homogeneous fields $(\mathbf{n}, \mathbf{c}) = (0, 0)$ and $(\mathbf{n}, \mathbf{c}) = (N, N/(1+N))$ are trivial solutions of the steady-state problem. Since these fields represent

densities of biochemical materials, the zero solution is obviously uninteresting. We consider in what follows the steady-state problem around its second homogeneous solution, that is after performing the shift

$$\mathbf{n} \leftarrow \mathbf{n} - N \quad \text{and} \quad \mathbf{c} \leftarrow \mathbf{c} - \frac{N}{1+N}, \tag{7.6}$$

and remember the natural constraints $\mathbf{n}, \mathbf{c} \geqslant 0$ (on the untranslated fields).

7.2.2 Geometrical modelling

In contrast to the works of Maini *et al.* (1991), Murray (2003), Winters *et al.* (1990), we consider curved domains, thus adding to the problem the effect of the curvature due to the domain geometry. More precisely, we will focus on idealized 2-D geometries that are given in terms of curvilinear coordinates for which the shape of the gradient and Laplace-Beltrami operators, ∇ and Δ, are well-known (see Arfken (1970, Chapter 2), or Abramowitz and Stegun (1964, Chapter 21)). Hence, we consider here cylindrical and prolate spheroidal domains. Boundary conditions for the system (7.5) are either periodic where the domain is closed, or zero-flux on the boundary.

7.2.2.1 Cylinder surface

Let us first consider a cylinder whose axis coincides with the z-axis. We may then use the cylindrical coordinates (R, θ, z), defined from Cartesian coordinates (x, y, z) by

$$\begin{cases} x = R\cos\theta \\ y = R\sin\theta \\ z = z \end{cases} \tag{7.7}$$

where the cylinder radius R is nonnegative, and where the azimuthal angle θ can fall anywhere on a full circle, between 0 and 2π. As the radius R is fixed on that cylinder, the Laplace-Beltrami operator reduces to

$$\Delta f = f_{zz} + R^{-2} f_{\theta\theta}, \tag{7.8}$$

and the derivative terms in (7.5a) are then

$$\begin{aligned} D\Delta\mathbf{n} - \alpha\nabla\cdot(\mathbf{n}\nabla\mathbf{c}) = \\ [D\mathbf{n}_{zz} - \alpha(\mathbf{n}\mathbf{c}_{zz} + \mathbf{n}_z\mathbf{c}_z)] + R^{-2}[D\mathbf{n}_{\theta\theta} - \alpha(\mathbf{n}\mathbf{c}_{\theta\theta} + \mathbf{n}_\theta\mathbf{c}_\theta)] \end{aligned} \tag{7.9}$$

where the subscript denotes derivation with respect to the corresponding variables: $\mathbf{n}_z = \partial\mathbf{n}/\partial z$ and $\mathbf{n}_{zz} = \partial^2\mathbf{n}/\partial z^2$. With these coordinates, we can define two interesting idealized geometries, a *full cylinder* where θ lies in $[0, 2\pi)$ and a *truncated cylinder* where θ lies in $(0, \theta_{\max})$ ($\theta_{\max} < 2\pi$). In the former case, the

boundary conditions are periodic in the θ direction and zero-flux on the edges $z = 0$ and $z = z_{\max}$, while in the latter the boundary conditions are zero-flux on all edges: $\theta = 0$, $\theta = \theta_{\max}$, $z = 0$ and $z = z_{\max}$.

Note that this surface appears just like a plane with special boundary conditions.

7.2.2.2 Prolate spheroid

We then consider *spheroids* (that is ellipsoids with two equal semi-diameters) centred at the origin and whose symmetry axes coincide with those of the Cartesian system. As the oblate case is similar, we only consider here the case of *prolate spheroids*, whose polar axis (aligned with the z-axis) is greater than their equatorial diameter. The associated prolate spheroidal coordinates (ξ, θ, ϕ) are related to the Cartesian coordinates (x, y, z) by the relations

$$\begin{cases} x = a \sinh \xi \sin \phi \cos \theta, \\ y = a \sinh \xi \sin \phi \sin \theta, \\ z = a \cosh \xi \cos \phi, \end{cases} \tag{7.10}$$

for a constant scale factor $a > 0$, and where $\xi \geqslant 0$, the azimuthal angle θ is in $[0, 2\pi)$ and the inclination angle ϕ is in $[0, \pi]$. Note that ξ is equal to $\operatorname{arctanh} \epsilon^{-1} = \log(\epsilon^{-1} + \sqrt{\epsilon^{-2} - 1})$, where ϵ is the eccentricity of any ellipsoidal section of the spheroid through the poles. Observe also that these coordinates present singularities at the poles (when $\phi = 0$ or π). Consequently, differential operators in these coordinates are not well-defined at these poles.

Fixing the variable ξ to a constant value determines a prolate spheroid of a particular shape. Taking into account this constraint, the Laplace-Beltrami operator reads in the prolate spheroidal coordinates as

$$\Delta f = \frac{f_{\phi\phi} + \cot \phi \, f_\phi}{a^2 \left(\sinh^2 \xi + \sin^2 \phi \right)} + \frac{f_{\theta\theta}}{a^2 \sinh^2 \xi \sin^2 \phi}, \tag{7.11}$$

and we also have that

$$\nabla \cdot (g \nabla f) = \frac{g(f_{\phi\phi} + \cot \phi \, f_\phi) + f_\phi g_\phi}{a^2 \left(\sinh^2 \xi + \sin^2 \phi \right)} + \frac{g f_{\theta\theta} + f_\theta g_\theta}{a^2 \sinh^2 \xi \sin^2 \phi}, \tag{7.12}$$

where the same convention for partial derivatives as above is used. The derivative terms in (7.5a) are then

$$\begin{aligned} D\Delta \mathbf{n} - \alpha \nabla \cdot (\mathbf{n} \nabla \mathbf{c}) = {} & \frac{[D\mathbf{n}_{\phi\phi} - \alpha(\mathbf{n}\mathbf{c}_{\phi\phi} + \mathbf{n}_\phi \mathbf{c}_\phi)] + \cot \phi [D\mathbf{n}_\phi - \alpha \mathbf{n}\mathbf{c}_\phi]}{a^2 \left(\sinh^2 \xi + \sin^2 \phi \right)} \\ & + \frac{D\mathbf{n}_{\theta\theta} - \alpha(\mathbf{n}\mathbf{c}_{\theta\theta} + \mathbf{n}_\theta \mathbf{c}_\theta)}{a^2 \sinh^2 \xi \sin^2 \phi}. \end{aligned} \tag{7.13}$$

7.2.3 Discretization

The geometric domain $\mathcal{D}$ is discretized using a homogeneous mesh in the general contravariant curvilinear coordinates (q^1, q^2). More precisely, assuming that the geometric domain is represented as the compact rectangle $[0, q^1_{\max}] \times [0, q^2_{\max}]$, it is then split in N_1 intervals following coordinate q^1, and in N_2 intervals following coordinate q^2, yielding a mesh with step sizes $h_i = q^i_{\max}/N_i$ (for $i = 1, 2$), and such that the grid points are labelled by the coordinates (q^1_i, q^2_j) with

$$q^1_i = ih_1 \quad \text{and} \quad q^2_j = jh_2 \tag{7.14}$$

for $i = 0, \ldots, N_1$ and $j = 0, \ldots, N_2$. The unknown fields $\mathbf{n}$ and $\mathbf{c}$ may be approximated on those grid points, and we note

$$n_{ij} = \mathbf{n}(q^1_i, q^2_j) \quad \text{and} \quad c_{ij} = \mathbf{c}(q^1_i, q^2_j). \tag{7.15}$$

These components are gathered in two vectors n and c, respectively, using the lexicographical order (see the second example in Subsection 3.1.1).

On this grid, the partial derivatives may be approximated using central finite-difference (see Section 4.1); for instance,

$$\mathbf{n}_{q^1}(q^1_i, q^2_j) \simeq \frac{n_{i+1,j} - n_{i-i,j}}{2h_1}, \tag{7.16}$$

and

$$\mathbf{n}_{q^1q^1}(q^1_i, q^2_j) \simeq \frac{n_{i+1,j} - 2n_{i,j} + n_{i-1,j}}{h_1^2}. \tag{7.17}$$

An approximation of the left-hand sides of equations (7.5) at each grid point (q^1_i, q^2_j) may then be computed using the expression of the derivative terms (on the particular geometry under consideration) and the finite-difference approximations of partial derivatives, yielding two residuals $F^n_{i,j}$ and $F^c_{i,j}$ on equations (7.5a) and (7.5b), respectively. We then aim to make these residuals equal to zero. Our discretized version of the steady-state problem (7.5) consists in the least-squares problem

$$\min_{n,c} F(n, c) = \frac{1}{N_1 N_2} \sum_{i=1}^{N_1} \sum_{j=1}^{N_2} [F^n_{i,j}(n, c)^2 + F^c_{i,j}(n, c)^2]. \tag{7.18}$$

Note that we multiply each term of this summation by $\sin^2 q^2_j$ for prolate spheroidal domains in order to avoid coordinate singularities ($q^2 = \phi$). This has however the drawback to reduce the accuracy required in the polar regions compared to the equatorial region.

Boundary conditions. — On the geometries presented in Subsection 7.2.2, the coordinate q^1 always represents the azimuthal angle θ. Except for the

truncated cylinder, the boundary conditions at $q^1 = 0$ and $q^1 = q^1_{\max}$ are periodic, meaning that

$$n_{0,j} = n_{N_1,j} \quad \text{and} \quad c_{0,j} = c_{N_1,j}, \tag{7.19}$$

for $j = 0, \ldots, N_2$. For the truncated cylinder, the normal component of the gradient of the fields must be zero on these boundaries. This Neumann condition is enforced by setting

$$n_{0,j} = n_{-1,j}, n_{N_1,j} = n_{N_1+1,j}, c_{0,j} = c_{-1,j}, c_{N_1,j} = c_{N_1+1,j}, \tag{7.20}$$

for $j = 0, \ldots, N_2$. On the other edges of the full and truncated cylinders, the corresponding Neumann conditions are used. On the spheroids, we use the polar joint conditions

$$n_{i_1,0} = n_{i_2,0}, \quad n_{i_1,q^2_{\max}} = n_{i_2,q^2_{\max}}, \quad c_{i_1,0} = c_{i_2,0}, \quad c_{i_1,q^2_{\max}} = c_{i_2,q^2_{\max}}, \tag{7.21}$$

for any $i_1, i_2 \in \{0, \ldots, N_1\}$. No variable is used in fact for the fields $\mathbf{n}$ and $\mathbf{c}$, at the North and South poles; the values at these points are fixed *a priori* at zero. This choice seems reasonable since it corresponds to the homogeneous equilibrium solution whereas no information is available at the poles because of the coordinate singularities.

7.3 Numerical pattern generation

7.3.1 Implementation

Let us now turn to the numerical solution of the optimization problem (7.18) arising from the finite-difference discretization of the steady-state model (7.5). We used the RMTR package (see Section 3.2) to solve numerically this problem. The sparse Hessian of the objective function was computed with the LTS package (see Subsection 5.1.1).

Except for the stopping criterion thresholds, the default parameters were used in the RMTR package. The algorithm was stopped as soon as one of the following criteria was satisfied:

- the objective function $F(n, c)$ was below 10^{-8};
- the criticality measure (3.25) was below 10^{-4};
- significant numerical noise was detected, in the sense that the model reduction is relatively close to machine precision (a feature of the RMTR package).

The starting point was chosen as the homogeneous fields $n = c = 0.01$, in order not to artificially induce structures in the solution. The values of the

biochemical parameters were inspired by choices of Maini *et al.* (1991), Murray (2003), Winters *et al.* (1990), that is a ratio of diffusion coefficients $D = 0.25$, a cell density at equilibrium $N = 1.0$, a cell growth rate $r = 1.5$, and a scale factor $s = 1.0$. Finally, the grid size $N_1 \times N_2$ was taken as 288×161 for the full cylindrical case, as 257×193 for the truncated cylindrical case, and as 320×319 for the spheroidal case.

All codes were written in Fortran 95 and the experiments were mainly conducted on the cluster URBM–SYSDYN[1] at the University of Namur (14 nodes, each one with 2×4 cores at 2.8 GHz and 16 GB of RAM).

7.3.2 Generated patterns

We now exhibit some patterns that were produced by the RMTR code as solution of (7.18).

Patterns on the full cylinder. — Figures 7.1 and 7.2 show the results of the cell-chemotaxis model solution on a cylinder, for different choices of the chemotaxis parameter α, and of the cylinder radius R. More specifically, we display the discretized fields n and c, a 3-D representation of the discretized field n, as well as the residual

$$\sqrt{F^n_{i,j}(n,c)^2 + F^c_{i,j}(n,c)^2}. \tag{7.22}$$

The cylinder height is 10 in both cases.

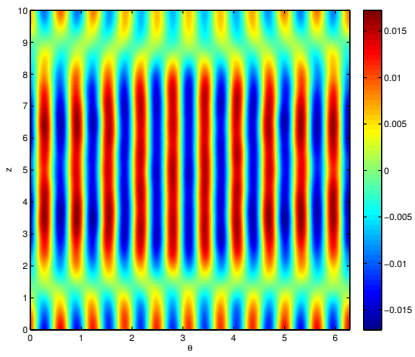
Patterns on the truncated cylinder. — Figure 7.3 on page 163 shows the results of the cell-chemotaxis model solution on a truncated cylinder, for a particular choice of parameters. The subfigures display the same quantities as for the full cylinder. In this example, the cylinder height is 3 and its radius is 1. The cylinder surface covers an azimuthal angle of 4 radians.

Patterns on the prolate spheroid. — Figures 7.4 and 7.5, on pages 164–165, finally show the results of the cell-chemotaxis model solution on a prolate spheroid, for different choices of the chemotaxis parameter α, and of the spheroid eccentricity ϵ. The subfigures display the same quantities as with the previous geometries. The spheroid semi-major axis is 10 in both cases.

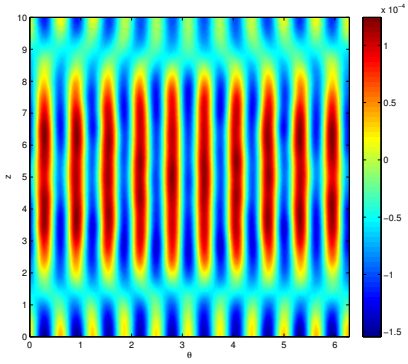
7.3.3 Performance of the RMTR method

We now discuss the performance of the RMTR method on this application. Let us first consider the solution of the cell-chemotaxis model (7.18) discretized on a cylinder ($a = 1.2$, $z_{\max} = 10$), with chemotaxis parameter fixed at $\alpha = 20$.

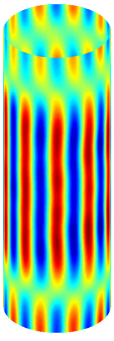
[1] URBM: Unité de recherche en biologie moléculaire; SYSDYN: Unité de recherche en systèmes dynamiques.



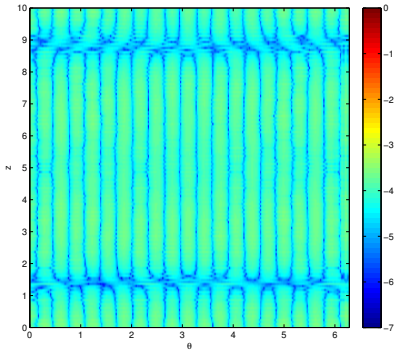
(a) *Cell density (flat view).*



(b) *Chemoattractant concentration.*

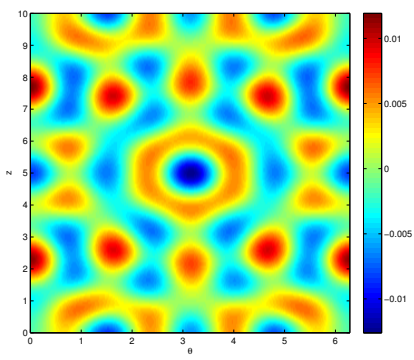


(c) *Cell density (3-D view).*

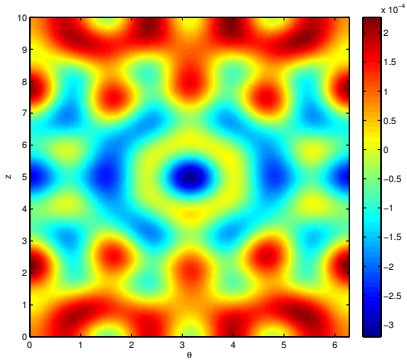


(d) *Residual (logarithmic colour scale).*

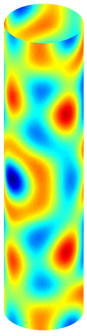
Figure 7.1 — *Solution of the cell-chemotaxis model ($\alpha = 40$) on a full cylinder ($R = 1.7$, $z_{\max} = 10$).*



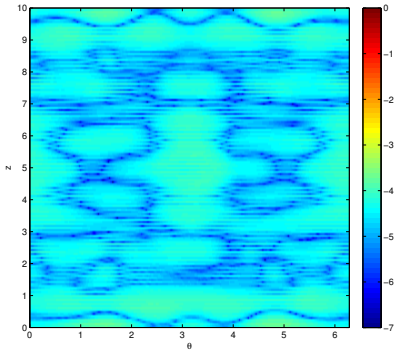
(a) *Cell density (flat view).*



(b) *Chemoattractant concentration.*

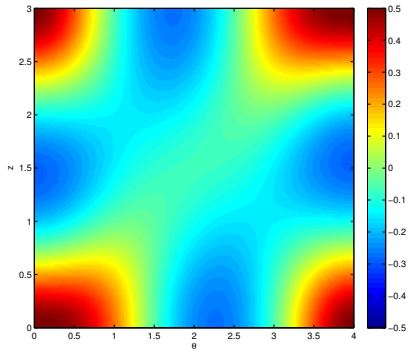


(c) *Cell density (3-D view).*

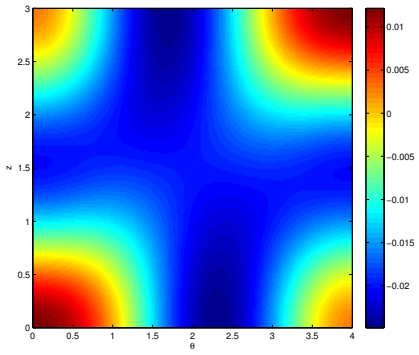


(d) *Residual (logarithmic colour scale).*

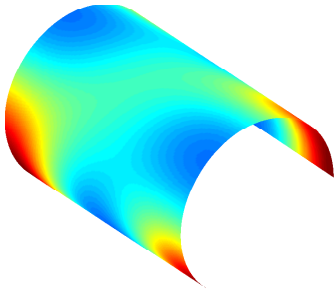
Figure 7.2 — *Solution of the cell-chemotaxis model ($\alpha = 20$) on a full cylinder ($R = 1.2$, $z_{\max} = 10$).*



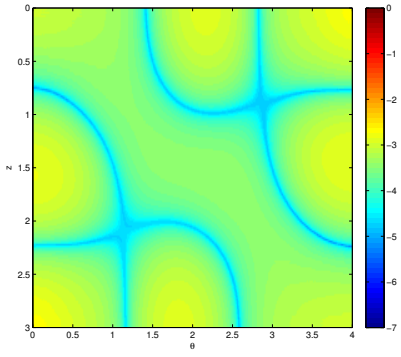
(a) *Cell density (flat view).*



(b) *Chemoattractant concentration.*

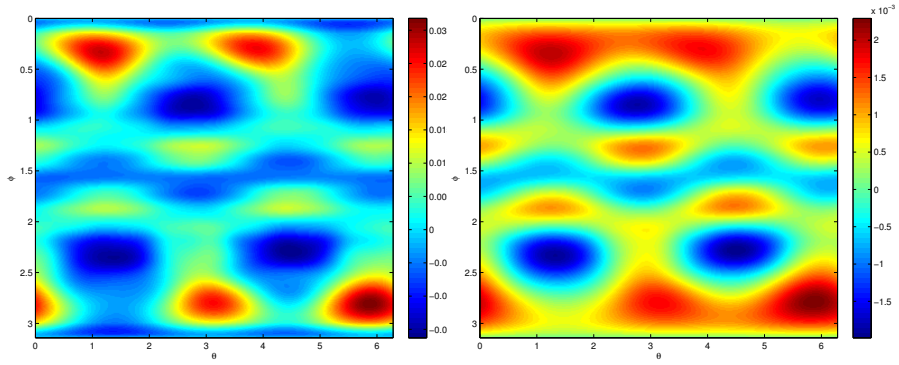


(c) *Cell density (3-D view).*



(d) *Residual (logarithmic colour scale).*

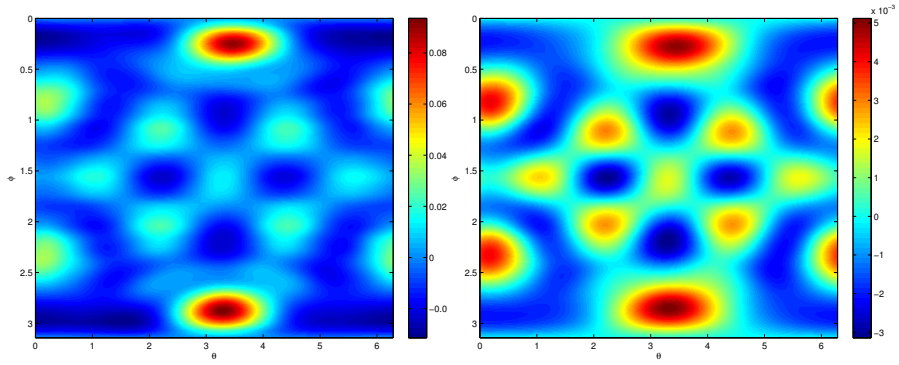
Figure 7.3 — *Solution of the cell-chemotaxis model ($\alpha = 14.5$) on a truncated cylinder ($R = 1$, $z_{\max} = 3$, $\theta_{\max} = 4$).*



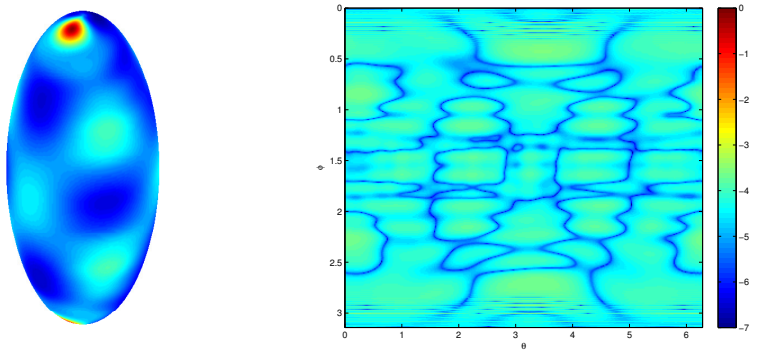
(a) *Cell density (flat view).* (b) *Chemoattractant concentration.*

(c) *Cell density (3-D view).* (d) *Residual (logarithmic colour scale).*

Figure 7.4 — *Solution of the cell-chemotaxis model ($\alpha = 15$) on a prolate spheroid ($a = 10$, $\epsilon = 0.95$).*



(a) *Cell density (flat view).* (b) *Chemoattractant concentration.*



(c) *Cell density (3-D view).* (d) *Residual (logarithmic colour scale).*

Figure 7.5 — *Solution of the cell-chemotaxis model ($\alpha = 15$) on a prolate spheroid ($a = 10$, $\epsilon = 0.9$).*

Level	4	5	6	Equiv.
PTCG	877	0	0	
Smoothing (SCM)	0	4,237	1,646	
Function evaluations	193	3,627	3,731	4,649.8
Gradient evaluations	5,575	2,641	5,185	6,193.7
Hessian approximations	174	21	115	131.1
Hessian reductions	0	136	115	
Prolongations	686	318	0	
Restrictions	0	18,357	7,637	

Table 7.1 — *Solving costs for discretized problem* (7.18) *on a cylinder* ($a = 1.2$, $z_{\max} = 10$, $\alpha = 20$)*, starting from homogeneous fields. Costs are reported level by level in columns labelled* 4 *to* 6*. In column* Equiv.*, we report the equivalent numbers of evaluations on the finest level.*

Three levels of discretization (numbered 4, 5 and 6) were used, and the mesh grid had 288×161 nodes at the fine level (level 6). Despite the potential slowing down of the RMTR method, coarser levels (1 to 3) were not used because they are usually too rough to represent interesting structures, leading in many cases to the zero homogeneous solution. Table 7.1 reports the main costs that occurred in the solution of this problem with the RMTR method, starting from the homogeneous fields set at 0.01. We therein indicate the number of times the Taylor's model is minimized by the PTCG or SCM methods (see Subsection 3.2.4), the number of function and gradient evaluations, and of Hessian approximations, the number of times the Hessian is reduced (to compute the Galerkin approximation at the coarser level), and finally the number of prolongations and restrictions. The algorithm terminated (because it encounters too much numerical noise) with a function value $F(n, c) = 1.050 \cdot 10^{-8}$ and criticality measure $6.44 \cdot 10^{-3}$ in 8,220 seconds.

The convergence process is notably influenced by the choice of the starting point. Taking for instance a Fourier mode $(3, 2)$ on the grid yields the results reported in Table 7.2. The algorithm terminates normally this time with a function value $F(n, c) = 9.988 \cdot 10^{-9}$ and criticality measure $5.10 \cdot 10^{-2}$ in 2,530 seconds.

Comparable results were observed in our experiments. The convergence of the RMTR method on the discretized cell-chemotaxis problem is therefore not so impressive, while still acceptable. It is in particular influenced by the starting point, and the value of the chemotaxis parameter, which brings more or less nonlinearity in the objective function F. We also observe that an important amount of work is performed at the finest level, meaning that multilevel methods would only be partially suited to solve this problem. Further experiments changing the value of the key parameter κ_χ (deciding whether a recursive it-

Level	4	5	6	Equiv.
PTCG	294	0	0	
Smoothing (SCM)	0	1,531	662	
f evaluations	133	1,635	1,484	1,901.1
g evaluations	3,643	1,399	1,812	2,389.4
Hessian approximations	114	19	37	48.9
Hessian reductions	0	56	37	
Prolongations	162	112	0	
Restrictions	0	6,501	3,005	

Table 7.2 — *Solving costs for discretized problem* (7.18) *on a cylinder* ($a = 1.2$, $z_{\max} = 10$, $\alpha = 20$)*, starting from Fourier mode (3,2). Costs are reported level by level in columns labelled* 4 *to* 6*. In column* Equiv.*, we report the equivalent numbers of evaluations on the finest level.*

eration is allowed) could perhaps improve the method performance. Changing the problem formulation would be another way to fasten the convergence. This could be done in several ways. First a better scaling of the variables could be performed if the expected (ranges of) values of the fields were known from the biological expertise (we indeed observe in our experiments an average ratio of about 100 between the cell and chemoattracant fields). Using finite elements could moreover allow to work on more general grids free of coordinate singularities.

7.4 Characterization of pattern complexity

Being able to solve the cell-chemotaxis model on several idealized geometries, we have soon faced a multitude of generated patterns to be analyzed. In order to be able to automatically distinguish patterns (potentially generated by a numerical method) and analyze their evolution with parameter variation, we need to characterize the complexity in their organization through quantitative estimators. Such tools are also necessary in the comparison of real patterns with numerical solutions, in particular to estimate the model adequacy to real data. We investigate two different families of methods. The first is based on the properties of Fourier transform of patterns (Subsection 7.4.1), while the second exploits random walks on the discrete graph of the discretization cells that constitutes a numerical pattern (Subsection 7.4.2).

The analyses were conducted using MATLAB® (using notably the `fft2` function for the Fourier transform), and we consider here only the patterns generated on the full cylinder for the (pigmentation) cell density. For the sake

of readability, we refer to Figure 7.1a as pattern A, and to Figure 7.2a as pattern B.

7.4.1 Fourier techniques

Fourier transform is a well-known tool for signal analysis. In this subsection, we therefore apply a 2-D discrete Fourier transform on the patterns, *i.e.* we decompose the pattern values n on a Fourier basis:

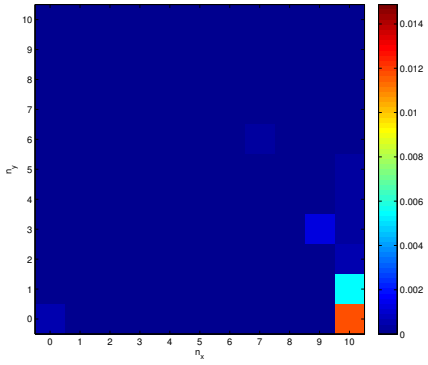
$$n_{ab} = \sum_{j=0}^{N_1} \sum_{k=0}^{N_2} f_{jk} \exp(-2iaj\pi/N_1) \exp(-2ibk\pi/N_2) \tag{7.23}$$

for $a = 0, \ldots, N_1$ and $b = 0, \ldots, N_2$, and where $i = \sqrt{-1}$. The intensities $|f_{jk}|$ of the Fourier modes may then be graphically represented as a 2-D image. However, this information may still be tough to interpret. We therefore perform a polar analysis of the frequencies found in the pattern. For this purpose, we first aggregate the intensities of modes (j, k) according to the value of $\lfloor \sqrt{j^2 + k^2} \rfloor$, yielding a radial power spectrum. These radial measures give information on the characteristic order of the structures present in the pattern. Secondly, we aggregate the intensities radially on the basis of $\lfloor M \arctan(k/j) \rfloor$ (where M is a parameter determining the number of classes). These angular measures now give information on the shape distribution of these structures, in the sense that they indicate whether the structures are rather round or vertically/horizontally elongated.

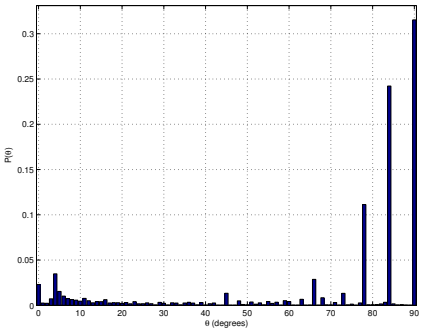
Figures 7.6 and 7.7 show the 2-D Fourier transform of patterns A and B, respectively, the corresponding angular and radial power spectra, as well as the cumulative radial power spectra. These latter graphs are used to measure the complexity of the patterns. A fast transition between the 0 and 1 values indeed indicates that few Fourier modes are present in the pattern. The slower this transition is, the more complex the pattern is (until white noise if the transition is linear).

In our first example, we observe on the radial power spectrum (Figure 7.6c) a narrow peak at value 10 (which corresponds mostly to the activated Fourier modes $(10, 0)$, $(10, 1)$, $(10, 2)$). Pattern A is accordingly not very complex and present 10 pairs of vertical stripes. The shape of the patches on the pattern is highly vertical as shown by the angular power spectrum (Figure 7.6b). Faint substructures present in the stripes are also detected on this graph that completes well the radial power spectrum.

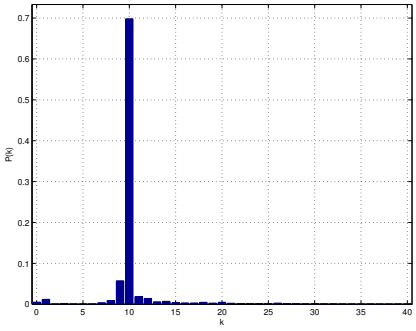
On the radial power spectrum of our second example (Figure 7.7c), we observe a larger region in which the main Fourier modes are gathered. This indicates an increased complexity of pattern B. We also observe a larger dispersion on the angular power spectrum (Figure 7.7b), and in particular the presence of rounder shapes and of a general horizontal background in pattern B.



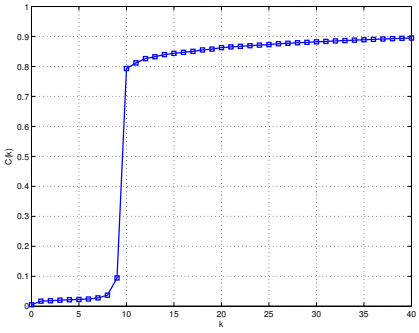
(a) *2-D Fourier transform.*



(b) *Angular power spectrum.*

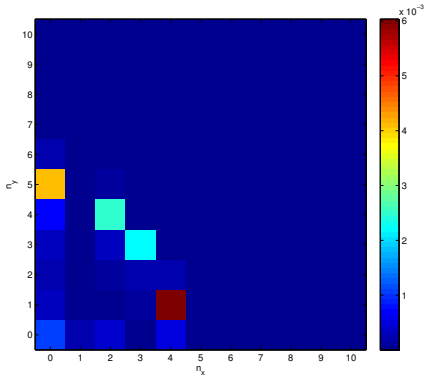


(c) *Radial power spectrum.*

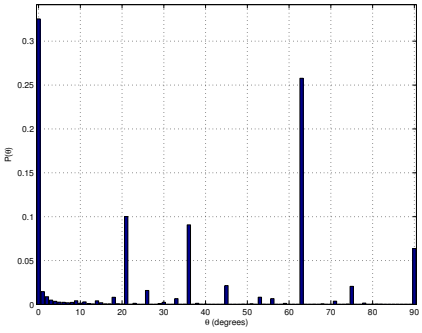


(d) *Cumulative radial power spectrum.*

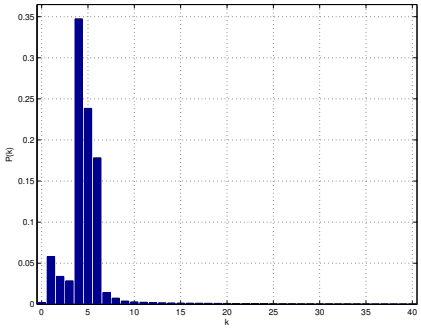
Figure 7.6 — *Fourier analysis of pattern* A *(Figure 7.1a).*



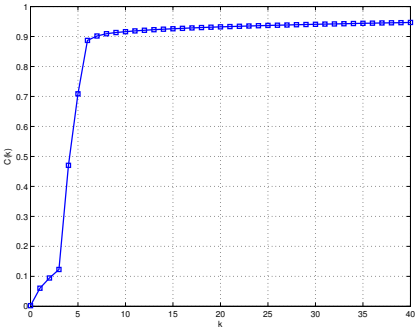
(a) *2-D Fourier transform.*



(b) *Angular power spectrum.*



(c) *Radial power spectrum.*



(d) *Cumulative radial power spectrum.*

Figure 7.7 — *Fourier analysis of the pattern* B *(Figure 7.2a).*

7.4.2 Random walks on discrete graphs

Our second tool for classifying patterns comes from graph theory, strongly inspired by the work of Delvenne, Yaliraki and Barahona (2009) on graph clustering. It considers the discretization grid as a graph, whose vertices are the grid points and whose edges connect each vertex to its North, South, West and East neighbours (taking the boundary conditions into account). We denote by $A \in \{0,1\}^{n\times n}$ the adjacency matrix of this graph. The components of the vector $d \in \mathbb{N}^n$ hold the degree of each node, and we set $D = \mathrm{diag}(d)$. We also define the vector π as $d/\sum_{i=1}^{n} d_i$ (which gives the degree distribution), and the matrix $\Pi = \mathrm{diag}(\pi)$.

On this graph, we build a *random walk* (see for instance Lovász, 1993), *i.e.* a Markov chain whose states are the vertices of the graph and in which the probability of leaving a vertex is uniformly distributed among the outgoing edges, that is with a transition probability $1/d_i$ for each edge. So the (normalized) probability vector p_t changes according to

$$p_{t+1} = p_t[D^{-1}A] = p_t M, \tag{7.24}$$

defining the transition matrix M. We consider that a real value α_i is assigned to each vertex i (in our case, the value n_i^* of the problem solution at that grid point). We then observe the signal produced along the random walk: it is a stationary random variable $(X_t)_{t\in\mathbb{N}}$ that consists of a sequence of α_i.

The patterns can then be characterized through this observable. In presence of fine structures, frequent variations of the signal are indeed expected along the random walk, while these changes should be less frequent if the pattern consists of visually large structures. Moreover, the speed at which transitions occur gives information on the sharpness of the edges. The time scale serves here as a resolution parameter for the determination of structures within the pattern. Escaping from a large structure should indeed take in average more time than to escape from a small one.

These phenomenons can be quantified through the autocovariance of the observable, that is the covariance of the signal against a time-shifted version of itself: $\mathrm{cov}[X_t, X_{t+\tau}] = \mathbb{E}[X_t X_{t+\tau}] - \mathbb{E}[X_t]^2$, where $\mathbb{E}$ denotes the expectation of a random variable (see for instance Glover, Jenkins and Doney, 2005). The autocovariance of X_t is then

$$\mathrm{cov}[X_t, X_{t+\tau}] = \alpha^T(\Pi M^\tau - \pi\pi^T)\alpha. \tag{7.25}$$

As this value does not depend on t (because the random variable is stationary), it is simply denoted by $\mathrm{cov}(\tau)$. This measure can also be normalized by dividing it by the variance (that is $\mathrm{cov}(0)$), yielding the autocorrelation

$$\rho(\tau) = \frac{\mathrm{cov}(\tau)}{\mathrm{cov}(0)}. \tag{7.26}$$

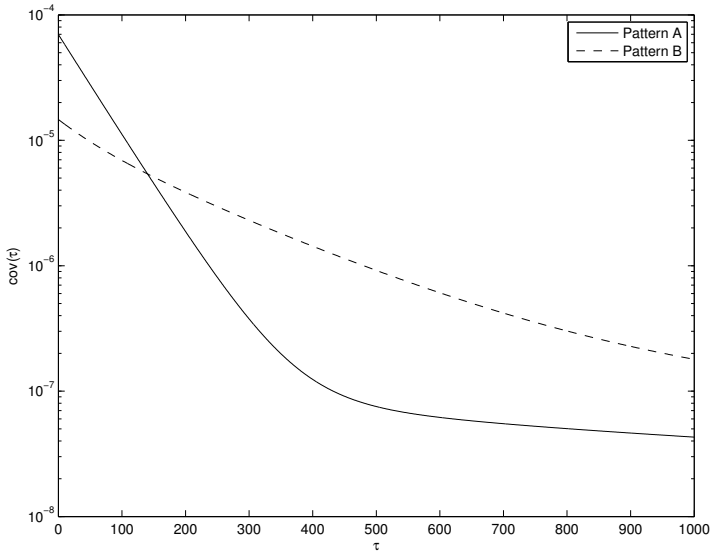


Figure 7.8 — *Autocovariance of the signal produced along a random walk on patterns* A *and* B *(logarithmic vertical scale).*

It indicates to what extent the next values of the signal depend on the current one. The decay of the autocovariance with the lag τ therefore reflects the importance of structures in the pattern.

In Figure 7.8, we display the autocovariance corresponding to patterns A and B, while the autocorrelation for these patterns are shown in Figure 7.9 on page 173. Figure 7.8 shows that pattern A has more contrast (its curve starts higher) than pattern B, which is a little more uniform. But as the curve for pattern A becomes dominated by that for pattern B, it means that the initial colour of the random walk is more quickly forgotten in the former pattern, and thus that its structures are smaller (at least in some direction). We also observe an elbow in the curve corresponding to pattern A. We connect that to the fact that the two different slopes correspond to the anisotropy of this pattern, with two distinct characteristic dimensions (a small horizontal width, but a large vertical height) of the patches in that figure. Along the horizontal direction, the random walker forgets quickly the original intensity, though it is however possible that it remains on the same vertical stripe, then retaining the information about the initial colour.

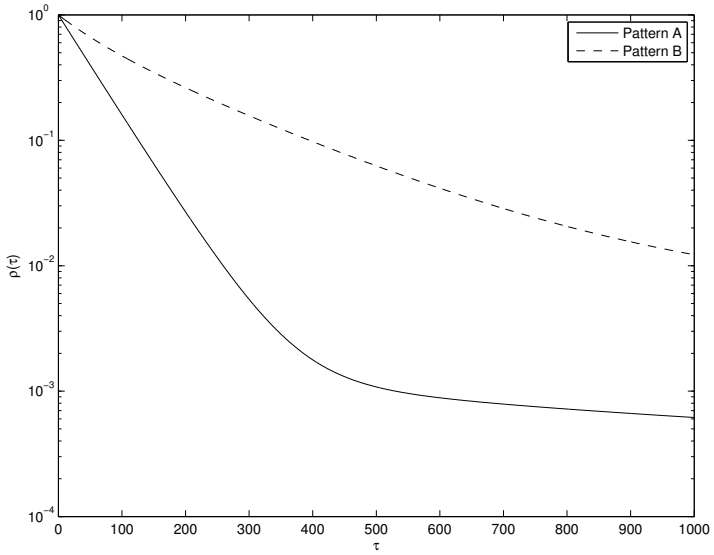


Figure 7.9 — *Autocorrelation of the signal produced along a random walk on patterns A and B (logarithmic vertical scale).*

7.5 Perspectives

The developments presented so far in this chapter are the beginnings of a broader study of the pigmentation pattern on snake skins. A first research direction consists in the improvement of the model, for instance by incorporating more complex biochemical interaction mechanisms, or by taking into account the existence of several pigmentation species. Complementarily, we should be able to deal with realistic geometric domains. Finite-element methods (see Gockenbach, 2006, for a good introduction to this topic) need then to be used to tackle the potentially complex domain on which the problem is posed. Finally, the temporal aspect of the problem should be reintegrated.

We now elaborate on how the finite-element development could be achieved, first by considering a finite-element formulation of the cell-chemotaxis model, and then by discussing its solution with the RMTR method.

7.5.1 Finite-element decomposition

Assume that the equations (7.5) are now posed on a bidimensional domain Ω partitioned in finite elements. More precisely, we consider a triangulation $\mathcal{T}_h$ of Ω consisting of triangle elements T_i, and of N_v vertices p_j. We are then looking for an approximation of the fields $\mathbf{n}$ and $\mathbf{c}$ that lie in $\mathcal{P}_h^{(1)}$, the space of piecewise linear functions on the triangulation $\mathcal{T}_h$. This is a vector space

of dimension N_v, which has a basis $\{\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_{N_v}\}$, where $\boldsymbol{\phi}_i(p_j) = \delta_{ij}$ for $i, j = 1, \ldots, N_v$. So we may decompose the fields $\mathbf{n}$ and $\mathbf{c}$ on the basis of $\mathcal{P}_h^{(1)}$, yielding

$$\mathbf{n} = \sum_{j=1}^{N_v} n_j \boldsymbol{\phi}_j \quad \text{and} \quad \mathbf{c} = \sum_{j=1}^{N_v} c_j \boldsymbol{\phi}_j.$$

with the vectors n and c of $\mathbb{R}^{N_v}$ to be determined. These approximate fields are required to satisfy the weak version of equations (7.5), that is

$$I^n = \int_\Omega \left[D\Delta\mathbf{n} - \alpha\nabla\cdot(\mathbf{n}\nabla\mathbf{c}) + sr\mathbf{n}(N-\mathbf{n})\right]\mathbf{v}dS = 0 \tag{7.27a}$$

$$I^c = \int_\Omega \left[\Delta\mathbf{c} + s\left(\frac{\mathbf{n}}{1+\mathbf{n}} - \mathbf{c}\right)\right]\mathbf{v}dS = 0 \tag{7.27b}$$

for some set of test functions $\mathbf{v}$. Applying Stokes' theorem (see for instance Nakahara, 2003) yields

$$I^n = \int_{\partial\Omega} \langle \mathbf{v}(D\nabla\mathbf{n} - \alpha\mathbf{n}\nabla\mathbf{c}), u\rangle \, d\ell - \int_\Omega \left[(D\nabla\mathbf{n} - \alpha\mathbf{n}\nabla\mathbf{c})\cdot\nabla\mathbf{v} - sr\mathbf{n}(N-\mathbf{n})\mathbf{v}\right] dS, \tag{7.28}$$

and

$$I^c = \int_{\partial\Omega} \langle \mathbf{v}\nabla\mathbf{c}, u\rangle \, d\ell - \int_\Omega \left[\nabla\mathbf{c}\cdot\nabla\mathbf{v} - s\left(\frac{\mathbf{n}}{1+\mathbf{n}} - \mathbf{c}\right)\mathbf{v}\right] dS, \tag{7.29}$$

where u is a unit normal vector to the curve $\partial\Omega$. Observe that the first integral in both right-hand sides is zero by assumption (7.4). Using the Galerkin approach, the set of test functions $\mathbf{v}$ is chosen as $\mathcal{P}_h^{(1)}$ and we thus only need to test for the basis functions $\boldsymbol{\phi}_i$. Hence, using the decomposition of $\mathbf{n}$ and $\mathbf{c}$ on this basis, we obtain

$$\int_\Omega \left[\sum_j n_j \left(D\nabla\boldsymbol{\phi}_j\cdot\nabla\boldsymbol{\phi}_i - srN\boldsymbol{\phi}_j\boldsymbol{\phi}_i\right) - \sum_{j,k} n_k\left(c_j\alpha\boldsymbol{\phi}_k\nabla\boldsymbol{\phi}_j\cdot\nabla\boldsymbol{\phi}_i - n_j\boldsymbol{\phi}_i\boldsymbol{\phi}_j\boldsymbol{\phi}_k\right)\right] dS = 0,$$

and

$$\int_\Omega \left[\sum_j \left(c_j\nabla\boldsymbol{\phi}_j\cdot\nabla\boldsymbol{\phi}_i + (c_j - n_j)s\boldsymbol{\phi}_j\boldsymbol{\phi}_i\right) + \sum_{j,k} n_k c_j\left(\boldsymbol{\phi}_k\nabla\boldsymbol{\phi}_j\cdot\nabla\boldsymbol{\phi}_i + s\boldsymbol{\phi}_k\boldsymbol{\phi}_j\boldsymbol{\phi}_i\right)\right] dS = 0,$$

where the second equation has been multiplied by $(1 + \mathbf{n})$. These equations can be rewritten in a polynomial form

$$I_i^n(n,c) = \sum_j n_j(DK_{ij} - srNM_{ij}) - \sum_{j,k} n_k(c_j \alpha J_{ijk} - n_j L_{ijk}) = 0, \quad (7.30a)$$

$$I_i^c(n,c) = \sum_j (c_j K_{ij} + (c_j - n_j) s M_{ij}) + \sum_{j,k} n_k c_j (J_{ijk} + s L_{ijk}) = 0, \quad (7.30b)$$

where the symmetric stiffness matrix K and mass matrix M are defined by

$$K_{ij} = \int_\Omega \nabla \boldsymbol{\phi}_i \cdot \nabla \boldsymbol{\phi}_j dS \quad \text{and} \quad M_{ij} = \int_\Omega \boldsymbol{\phi}_i \boldsymbol{\phi}_j dS,$$

and where we also use the tensor J and L defined by

$$J_{ijk} = \int_\Omega (\nabla \boldsymbol{\phi}_i \cdot \nabla \boldsymbol{\phi}_j) \boldsymbol{\phi}_k dS \quad \text{and} \quad L_{ijk} = \int_\Omega \boldsymbol{\phi}_i \boldsymbol{\phi}_j \boldsymbol{\phi}_k dS.$$

Note that the entries K_{ij} and M_{ij} are nonzero only if points p_i and p_j are linked by an edge of the triangulation $\mathcal{T}_h$; similarly, the entries J_{ijk} and L_{ijk} are nonzero only when the points p_i, p_j and p_k lie on a face of the triangulation $\mathcal{T}_h$.

Working with the finite-element basis functions. — In what follows, we denote by $T(\alpha : \beta : \gamma)$ the barycentre of the triangle T with weights $(\alpha : \beta : \gamma)$. As the basis functions $\boldsymbol{\phi}_i$ are piecewise linear, their gradient is piecewise constant. We may therefore compute the matrices K and M, and the tensors J and L using Gauss quadrature of appropriate order (see Sections 7.1 and 8.1 of Gockenbach, 2006). The entries K_{ij} are thus determined with the Gauss quadrature of order 0:

$$K_{ij} = \sum_k |T_k| [\nabla \boldsymbol{\phi}_i \cdot \nabla \boldsymbol{\phi}_j]_{T_k},$$

where the summation takes place on every face T_k (whose area is denoted by $|T_k|$) that contains both points p_i and p_j (there are two such faces if p_i and p_j are distinct interior points). We compute J_{ijk} with the Gauss quadrature of order 1:

$$J_{ijk} = |T| [(\nabla \boldsymbol{\phi}_i \cdot \nabla \boldsymbol{\phi}_j) \phi_k]_{T(1:1:1)},$$

where T is the face with vertices p_i, p_j, p_k. We compute M_{ij} with the Gauss quadrature of order 2:

$$M_{ij} = \frac{1}{3} \sum_k \sum_{cycl} |T_k| [\boldsymbol{\phi}_i \boldsymbol{\phi}_j]_{T_k(1:1:0)},$$

where the summation over k includes the same faces as for K_{ij}, and where the cyclic summation takes place on the barycentric coordinates. Finally, we

compute L_{ijk} with the Gauss quadrature of order 3:

$$L_{ijk} = -\frac{9}{16}|T|[\boldsymbol{\phi}_i\boldsymbol{\phi}_j\boldsymbol{\phi}_k]_{T(1:1:1)} + \frac{25}{48}\sum_{cycl}|T|[\boldsymbol{\phi}_i\boldsymbol{\phi}_j\boldsymbol{\phi}_k]_{T(3:1:1)},$$

where T is the face with vertices p_i, p_j, p_k, and where the cyclic summation takes again place on the barycentric coordinates. The gradient of each basis function $\boldsymbol{\phi}_i$ is zero on every element T_ℓ that does not contain the vertex p_i. Otherwise, it is a vector in the plane determined by $T_\ell = (p_i, p_j, p_k)$ (because of (7.4)) and orthogonal to the opposite edge $[p_j, p_k]$ (because this line is a level curve of $\boldsymbol{\phi}_i$). If we denote the foot of the altitude of T_ℓ through p_i by $\hat{p}_i^\ell$, then we clearly have

$$[\nabla\boldsymbol{\phi}_i]_{T_\ell} = \frac{p_i - \hat{p}_i^\ell}{\|p_i - \hat{p}_i^\ell\|^2}. \tag{7.31}$$

This new point $\hat{p}_i^\ell$ can be numerically computed as

$$\hat{p}_i^\ell = T_\ell(0 : d_{jk}^2 + d_{ik}^2 - d_{ij}^2 : d_{jk}^2 + d_{ik}^2 - d_{ij}^2), \tag{7.32}$$

where $d_{ab} \overset{\text{def}}{=} \|p_a - p_b\|$ for every $a, b = 1, \ldots, N_v$ (see for instance Lalesco, 1952).

Least-squares formulation. — We then may solve equations (7.30) by minimizing the sum of the squared residuals:

$$F(n, c) = \frac{1}{2}\sum_{i=1}^{N_v}\left[I_i^n(n, c)^2 + I_i^c(n, c)^2\right].$$

Its gradient may be computed analytically:

$$\frac{\partial F}{\partial n_\ell} = \sum_{i=1}^{N_v}\left(I_i^n\left[DK_{i\ell} - srNM_{i\ell} - \sum_j(c_j\alpha J_{ij\ell} - n_j(L_{i\ell j} + L_{ij\ell}))\right] - I_i^c\left[sM_{i\ell} - \sum_j c_j(J_{ij\ell} + sL_{ij\ell})\right]\right)$$

and

$$\frac{\partial F}{\partial c_\ell} = \sum_{i=1}^{N_v}\left(I_i^n\left[\sum_j n_j\alpha J_{i\ell j}\right] + I_i^c\left[K_{i\ell} + sM_{i\ell} + \sum_j n_j(J_{i\ell j} + sL_{i\ell j})\right]\right),$$

for $\ell = 1, \ldots, N_v$. This completes the description of the cell-chemotaxis model in the finite-element formalism.

7.5.2 Multilevel methods on finite-element domains

We now discuss how the RMTR method could be adapted to tackle problems defined on finite-element domains. If we use the Galerkin approximation to define the coarse model in the RMTR method, the transfer operators are the sole elements of the multilevel hierarchy that are needed. In the linear case, some multigrid methods, called *algebraic multigrid* methods, are able to guess these operators directly from the matrix of the linear system, which somehow measures the connectivity between the variables (see Trottenberg *et al.*, 2001). In the nonlinear case that we consider, we could use periodically the linear strategies on the current Hessian matrix to define the transfer operators for the following iterations. We however propose here to define an *a priori* hierarchy of coarsened versions of the finite-element decomposition using a mesh simplification tool.

Using an iterative mesh simplification tool that removes vertices one by one, we may store the neighbours vertices of the removed one and its barycentric coordinates in the face determined by these neighbours, yielding the construction of transfer operators. An appropriate mesh coarsening is for instance the quadric-based edge-collapse simplification method of Garland (1999) (see also Heckbert and Garland, 1999) for which an efficient implementation exists in C++ (QSlim 2.1). Several algorithmic choices are proposed by this method (like ensuring that every coarse node is also a fine node, or considering known normal vectors to the surface), and still need to be examined, as well as the definition of coarsening levels that would constitute the multilevel hierarchy.

Chapter 8

The RMTR and LTS packages

This last chapter is devoted to the software engineering that took place during this thesis. We only elaborate here on the two software packages that are made publicly available through the GALAHAD[1] library of Gould *et al.* (2003*b*), namely the RMTR and LTS packages. As their name indicates, the former implements the RMTR method presented in Section 3.2, while the latter implements the LTS method described in Subsection 5.1.1.

The development of the RMTR code was mainly conducted by Dimitri Tomanos. Our contribution to this package consists in its adaptation for use with the LTS package, the improved facilities to define the grids on which the multilevel problems are posed and the transfer operators between them (notably by providing the required routines for common cases), the fixing of some bugs and the addition of some minor features. The code and documentation maintenance has also been carried out since mid-2009. A version of the code accepting finite-element decompositions is still under development (see Section 7.5).

We developed concomitantly the LTS code to approximate the Hessian in the context of multilevel problems. The lower-triangular substitution (LTS) method was implemented in this package, since it appears as the most efficient and robust one from the experiments reported in Section 5.5.

We focus here on some particular aspects of these packages. First, we describe in Section 8.1 how the transfer operators are defined in RMTR. Section 8.2 explains the Hessian management in RMTR. In addition to the extensive description of the LTS method in Subsection 5.1.1, we present in Section 8.3 the predefined sparsity pattern for which optimal column grouping may be directly provided by LTS. For a complete description of these packages, we refer

[1] The library is available on the website `http://galahad.rl.ac.uk`. All use is subject to licence (see `http://galahad.rl.ac.uk/galahad-www/cou.html`). For any commercial application, a separate license must be signed.

the reader to the current specifications of RMTR in Appendix D, and of LTS in Appendix E. We however omit in both cases the description of the methods, since they have already been presented in the former chapters.

8.1 Transfer operators in RMTR

The RMTR algorithm needs information on the multilevel structure of the problem. The difficulty herein is that multilevel problems may be quite different, even in the restricted class of discretized optimization problems, depending for instance on the chosen discretization technique and domain topology. But, on the other hand, many common multilevel problems arise from discretizations of an underlying infinite-dimensional problem on regular geometric grids. To cover as much ground as possible with the RMTR package, we have therefore chosen a flexible two-pronged strategy: the user may either provide its own multilevel information, or use the package facilities for problems defined on some particular regular discretization grids.

The RMTR package contains indeed predefined operators for several hierarchy of rectangular grids, that we now characterize. The ratio of their mesh sizes at two consecutive levels is always 2 (the mesh size at the finer level being smaller than at the coarser). The user may choose the dimension of the geometric space on which the grids are posed, and the number of variable fields that are considered, that is the number of variables that are defined at each node of the grid. He may also decide whether linear or cubic interpolation is used to define the prolongation operator. The grids are built from the coarsest one to the finest one. The user must therefore specify the number of nodes in each direction of the rectangular grid at the coarsest level. From this information and using user-chosen recursion rules, the numbers of variables at the finer levels are determined and the transfer operators between consecutive levels are constructed.

The recursion rules are stated for 1-D grids, and may therefore vary in each dimension of the rectangular grids of interest. They describe how the grid nodes from two consecutive levels are related. Six possibilities are implemented in the RMTR package. In each case, the free coarse nodes are also considered as fine nodes, and fine nodes are added (in the middle) between every two consecutive free coarse nodes. We now list these rules indicating their differences, and illustrate them in Figure 8.1 on the preceding page.

`OPERATORS_INTERIOR:` the boundary nodes are free coarse nodes, so no further fine node is added compared to the basic disposition.

`OPERATORS_EXTERIOR:` the boundary nodes are fixed nodes (with value zero, for coherence when prolongating iteration steps and gradients), so two additional fine nodes are added considering the boundary nodes as virtual coarse nodes.

`OPERATORS_LEFT`: only the boundary node on the left is fixed (at zero), so one additional fine node is added considering that boundary node as virtual coarse node.

`OPERATORS_RIGHT`: the symmetric case of `OPERATORS_LEFT` with the other boundary

`OPERATORS_LEFT_PERIODIC`: the same case as `OPERATORS_LEFT`, except that the value at the virtual coarse node is no longer zero, but the value at the other (free) boundary node.

`OPERATORS_RIGHT_PERIODIC`: the symmetric case of `OPERATORS_LEFT_PERIODIC` with the other boundary.

An explicit description of the parameters controlling these features in the RMTR package is available in Subsection D.2.5.7.

8.2 Hessian management in RMTR

If the Taylor's model (3.22) is chosen at Step 1 of Algorithm 3.2 (outlining the RMTR method), we must define at Step 3 of that algorithm a symmetric matrix $B_{i,k}$ approximating the Hessian of the function $f_{i,k}$ at the current iterate $x_{i,k}$. In the RMTR package, the user may either provide its own subroutine to evaluate this Hessian matrix, or use the LTS package to approximate it.

In both cases, computing a model Hessian is commonly one of the expensive task in the algorithm. To try reducing this cost, its computation may be skipped at some iterations, following one of the two strategies implemented in the RMTR package. In the first one, the Hessian is reevaluated periodically with a user-supplied frequency. The second (default) strategy was proposed by Gratton *et al.* (2010*b*), and aims to avoid recomputing the Hessian when the gradient variations are still well predicted by the available Hessian. More specifically, the Hessian is recomputed at the beginning of iteration (i, k) (with $k > 0$) only when the preceding iteration was not successful enough in the sense that $\rho_{i,k-1} < \eta_H$ for some constant $\eta_H \in (0, 1)$, or when

$$\|g_{i,k} - g_{i,k-1} - H_{i,k-1}s_{i,k-1}\|_2 > \epsilon_H \|g_{i,k}\|_2 \tag{8.1}$$

or

$$\|g_{i,k} - g_{i,k-1} - H_{i,k-1}s_{i,k-1}\|_\infty > \tilde{\epsilon}_H, \tag{8.2}$$

for some constant $\epsilon_H, \tilde{\epsilon}_H > 0$. Otherwise, $H_{i,k} = H_{i,k-1}$.

8.3 Predefined sparsity pattern in LTS

Similarly to our observation about transfer operators, the Hessian sparsity patterns of problems arising from discretization on rectangular grids are often

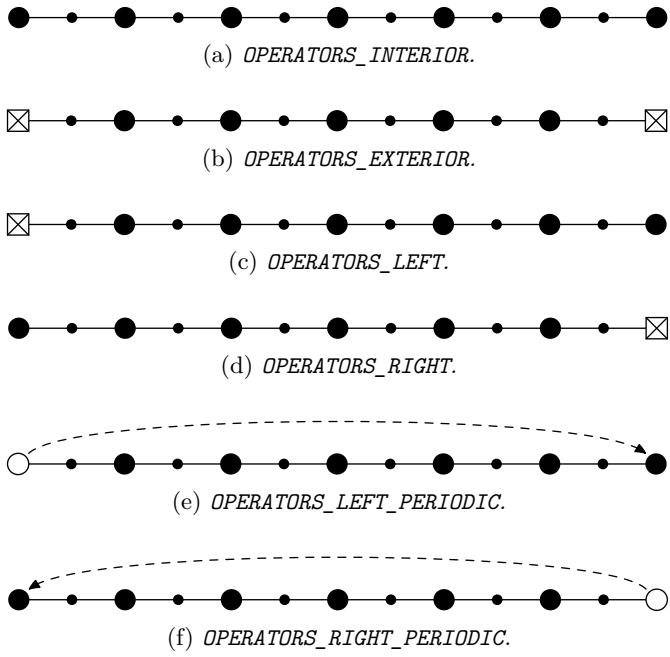
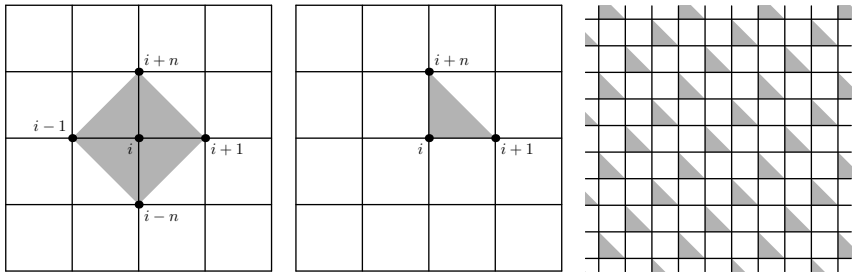


Figure 8.1 — *The possible recursion rules determining the (1-D) construction of the fine mesh from the coarse mesh. The large black circles represents vertices that belong to the coarse mesh (as well as the fine mesh), while the small ones represent vertices that are only present on the fine grid. The checked squares indicate zero Dirichlet boundary conditions, while the large white circles indicate periodic boundary condition (an arrow show the periodicity).*



(a) *Molecule.* (b) *Forward molecule.* (c) *Forward cover.*

Figure 8.2 — *Illustration of the predefined sparsity pattern 1.*

of few different types. We therefore aim to facilitate the LTS package use by avoiding the user the tedious task to implement well-known sparsity patterns. An optimal column grouping is then provided by the LTS package for each of these predefined Hessian sparsity patterns mostly thanks to Goldfarb and Toint (1984). In that case, no column grouping needs to be computed as explained in Steps 1 and 2 of Algorithm 5.1 on page 91.

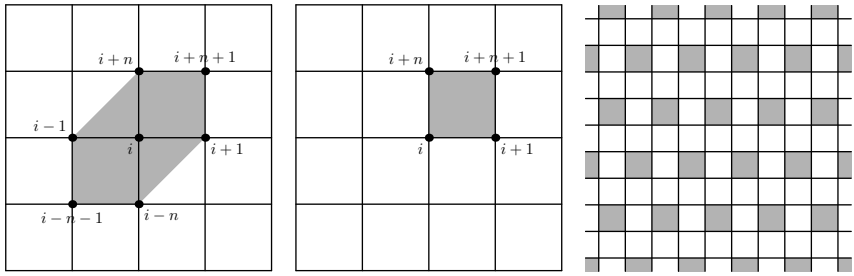
Currently, the predefined sparsity patterns are related to 2-D and 3-D problems posed on square grids. An adaptation to general rectangular grids could however be easily carried out in a future version of the package.

We now describe the predefined sparsity patterns and corresponding optimal column grouping (using the forward molecule and cover technique of Subsection 5.1.2):

Pattern 1. The underlying problem is posed on a 2-D grid with n^2 variables. The Hessian has 5 diagonals, and its potential nonzero entries on each row i are in columns $i-n$, $i-1$, i, $i+1$, $i+n$ as illustrated in Figure 8.2a. The optimal column grouping is given by the cover (with forward molecules) illustrated in Figure 8.2c.

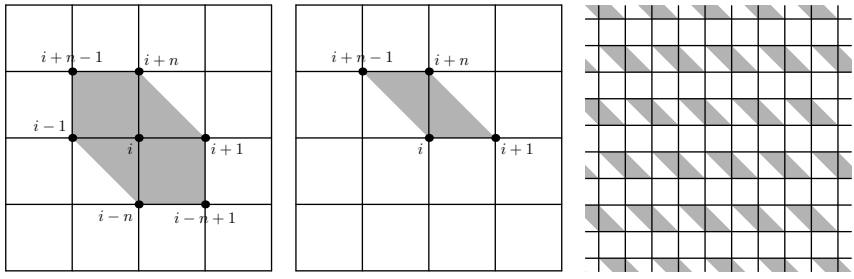
Pattern 2. The underlying problem is posed on a 2-D grid with n^2 variables. The Hessian has 7 diagonals, and its potential nonzero entries on each row i are in columns $i-n-1$, $i-n$, $i-1$, i, $i+1$, $i+n$, $i+n+1$ as illustrated in Figure 8.3a. The optimal column grouping is given by the cover (with forward molecules) illustrated in Figure 8.3c.

Pattern 3. The underlying problem is posed on a 2-D grid with n^2 variables. The Hessian has 7 diagonals, and its potential nonzero entries on each row i are in columns $i-n$, $i-n+1$ $i-1$, i, $i+1$, $i+n-1$, $i+n$ as illustrated in Figure 8.4a. The optimal column grouping is given by the cover (with forward molecules) illustrated in Figure 8.4c.



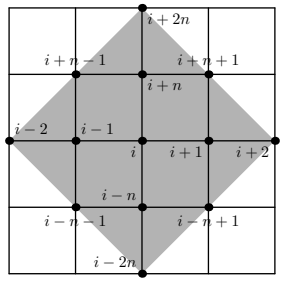
(a) *Molecule.* (b) *Forward molecule.* (c) *Forward cover.*

Figure 8.3 — *Illustration of the predefined sparsity pattern 2.*

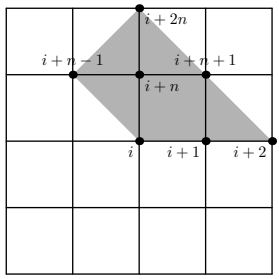


(a) *Molecule.* (b) *Forward molecule.* (c) *Forward cover.*

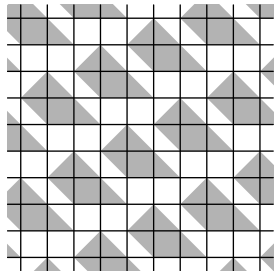
Figure 8.4 — *Illustration of the predefined sparsity pattern 3.*



(a) *Molecule.*

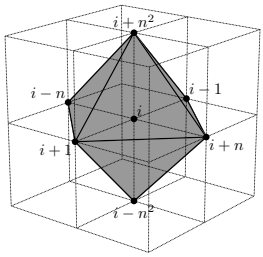


(b) *Forward molecule.*

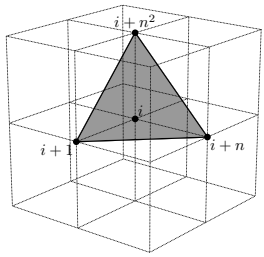


(c) *Forward cover.*

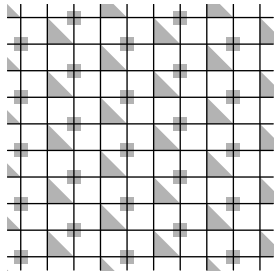
Figure 8.5 — *Illustration of the predefined sparsity pattern 4.*



(a) *Molecule.*

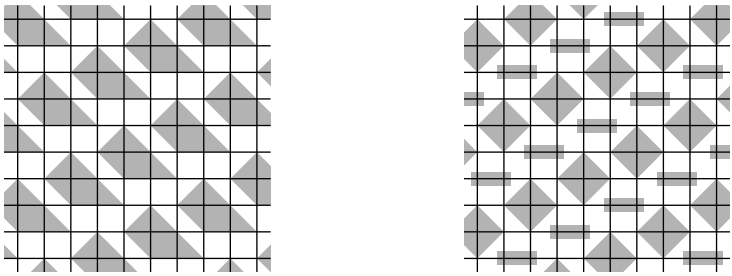


(b) *Forward molecule.*



(c) *Forward cover: horizontal layer.*

Figure 8.6 — *Illustration of the predefined sparsity pattern 5.*



(a) *Layer related to the first field.* (b) *Layer related to the second field.*

Figure 8.7 — *Illustration of the predefined sparsity pattern 6: two-layer cover.*

Pattern 4. The underlying problem is posed on a 2-D grid with n^2 variables. The Hessian has 13 diagonals, and its potential nonzero entries on each row i are in columns $i-2n$, $i-n-1$, $i-n$, $i-n+1$, $i-2$, $i-1$, i, $i+1$, $i+2$, $i+n-1$, $i+n$, $i+n+1$, $i+2n$ as illustrated in Figure 8.5a. The optimal column grouping is given by the cover (with forward molecules) illustrated in Figure 8.5c.

Pattern 5. The underlying problem is posed on a 3-D grid with n^3 variables. The Hessian has 7 diagonals, and its potential nonzero entries on each row i are in columns $i-n^2$, $i-n$, $i-1$, i, $i+1$, $i+n$, $i+n^2$ as illustrated in Figure 8.6a. The optimal column grouping is given by the cover (with forward molecules) illustrated in Figure 8.6c.

Pattern 6. The underlying problem is posed on a 2-D grid with two fields of variable, and $2n^2$ variables in total. The Hessian has the following block structure:

$$\left(\begin{array}{c|c} \text{Pattern 4} & \text{Pattern 1} \\ \hline \text{Pattern 1} & \text{Simple diagonal} \end{array}\right)$$

The optimal column grouping is given by the two-layer cover illustrated in Figure 8.7.

Conclusion and further research perspectives

The research work described in this thesis was concerned with the development and study of methods for solving nonlinear multilevel optimization problems, and more particularly with the design of Hessian approximations for such methods. The definition of these problems at different levels of accuracy, and the typical properties of their objective function (sparsity, partial separability) are indeed acknowledged as valuable and exploitable characteristics in the design of efficient methods.

After the introduction of basic concepts and tools in Chapter 1, we have presented the fundamentals of nonlinear optimization, and notably globalization strategies (linesearch and trust-region methods). We have moreover proposed a new method of the trust-region type, which differs from the classical approach by the use of the new model (instead of the current one) to determine the size of the new trust region. The convergence properties of this method have been discussed, and its numerical performance investigated.

In Chapter 3, we have described the basic ideas behind the multigrid method for linear systems of equations, and their adaptation to nonlinear optimization, especially in the framework of trust-region methods, leading to the RMTR method, whose convergence and numerical performance have been outlined. A few improvements to the software RMTR implementing this method have also been mentioned in Chapter 8, and recorded in its documentation (Appendix D).

The general subject of Hessian approximation have then been introduced in Chapter 4, focusing mainly on finite-difference methods and secant methods. These two last chapters have been the main motivation for this research work, described in the next chapters.

Approximating Hessians in unconstrained optimization arising from discretized problems

In Chapter 5, we have not directly exploited the hierarchy of descriptions that multilevel problems possess, but have instead been concerned with the typical sparsity and partial separability properties of their objective function. Two existing indefinite sparse Hessian approximation method have been reviewed, and notably the lower-triangular substitution (LTS) method of the finite-difference type. We have then proposed a secant Hessian update for partially separable functions and found out it is equivalent to a sparse secant update in a particular norm. An attempt to enforce positive definiteness of the Hessian approximation have also been conducted but with poor results. Numerical comparison between these methods have been carried out, and the LTS method stands out as the best choice amongst the considered algorithms, especially if an optimal column grouping is known. For this reason, this method was implemented in the LTS package of the GALAHAD optimization library, which is documented in Appendix E.

Multisecant L-BFGS methods

Chapter 6 considers this time the multilevel hierarchy to design Hessian approximation. We have been especially interested in limited-memory method. The L-BFGS method has therefore been presented before introducing the generation of multiple secant pairs at each iteration using a smoothing effect of the combination of the prolongation and restriction operators used to move between levels of the problem. Extensive numerical experiments have been conducted. They show an impressive improvement of the L-BFGS method when integrating the smoothed secant pairs in the design of a Hessian approximation. We have also attempted to better understand how the L-BFGS update deals with the secant pairs that it integrates. From this study, it appears that the L-BFGS update takes more care of the intrinsic quality of the secant pairs than of their position (except for the last one). We have finally proposed a new multisecant Hessian update that is able to give more or less importance to each secant pair through penalization of the secant equations. The matter of selecting appropriate penalization weights still remains open and calls for a deeper understanding of the L-BFGS method's behaviour.

Modelling of snake-skin pigmentation patterns

An application of the RMTR method using the LTS Hessian approximation to mathematical biology have been proposed in Chapter 7. We have indeed described the cell-chemotaxis model for snake-skin pigmentation patterns. The

PDEs modelling this problem have been discretized on curved domains (namely cylinders and spheroids) with finite-difference techniques, leading to a least-squares optimization problem. The performance of the solving process have then been discussed and seems to indicate that a better modelling of the original problem might be needed. Two complementary tools for the characterization of the generated patterns, namely Fourier analysis (with a radial-angular view) and random walks on discrete graphs, have been examined. We have finally discussed the possible development of a finite-element description of the problem that could tackle more realistic domains, and also reduce the nonlinearity of the objective function, hence hopefully fastening the solution of this problem.

Research perspectives

The multisecant methods still deserve more attention. Our understanding of the L-BFGS behaviour is indeed still sketchy. Moreover, the use of the multisecant L-BFGS method inside a trust-region method should be investigated, as well as the use of the smoothed pairs in other secant updates.

Another interesting subject of research would concern the definition of the transfer operators (and potentially the coarser levels themselves). Indeed, it is still unclear how a finite-element decomposition of a complex object given at the fine level should be coarsen to define simplified versions of the problem. More broadly, an adaptation of the (grid-free) algebraic multigrid techniques existing for linear systems would be a highly valuable development in the field of large-scale nonlinear optimization. In contrast to the linear case, in which the multigrid hierarchy is constructed once for all at the beginning, this expensive process could potentially be needed at each iteration of optimization methods, notably mitigating the interest in these techniques. Adaptive techniques may also be of interest. In that case, only some regions of the mesh are refined, depending on some criterion like the gradient norm, the curvature, or an *a posteriori* error estimator (see for instance Schmidt and Siebert, 2005, and Ziems and Ulbrich, 2008)

The introduction of more complex constraints would also be an asset to tackle practical applications. The integration of the multilevel philosophy within filter techniques (see Fletcher and Leyffer, 2002, and Fletcher, Leyffer and Toint, 2002) is an attractive lead to achieve this goal.

The mathematical study of snake-skin pigmentation patterns started in this thesis calls for further developments. On the one hand, the time dimension should be reintegrated in the problem, and more complex geometries might be considered. On the other hand, the characterization of pattern complexity remains an open field of research. Although the random walks are currently considered on the 2-D flat grid, we should probably take into account the geometry of the domain, by weighting the discretization graph with the real

distance between its nodes. One might also consider local characterization tools like, for instance, wavelets. Patterns are indeed not uniform distributed on snake skins, especially when one distinguishes its back, belly and head.

Summary of contributions

Our contributions are the study and the design of algorithms for nonlinear multilevel optimization, more specifically for the approximation of the Hessian matrix therein, as well as their application to a problem of mathematical biology, and the implementation of software. We summarize our contribution below.

- The retrospective trust-region method (Section 2.8) for solving unconstrained nonlinear optimization problems, its convergence theory and numerical experiments have been first presented by Bastin, Malmedy, Mouffe, Toint and Tomanos (2010).
- The partially separable secant Hessian updating procedure (Section 5.3), and its positive definite alter ego (Section 5.4), as well as the numerical comparison of Hessian approximation methods in the context of unconstrained nonlinear optimization problems arising from discretization (Section 5.5) are presented in Malmedy and Toint (2010).
- The numerical experiments on the L-BFGS method using secant pairs smoothed on the multilevel hierarchy are presented in Gratton, Malmedy and Toint (2010*a*). A new Hessian update penalizing multiple secant equations has also been introduced, but calls for more research to be of practical use.
- The modelling of snake-skin pigmentation patterns on curved domains, the numerical solution of the resulting least-squares problem, and the characterization of the generated patterns through Fourier analysis and random walks on discrete graphs are the subject of a paper in preparation by Carletti, Delvenne, Füzfa, Lambert, Malmedy and Toint (2010).
- The maintenance and improvement of the RMTR package, and the development of the LTS package have been carried out (Chapter 8, Appendices D and E). Both packages are included in GALAHAD, and a technical paper on this software is in preparation by Malmedy, Toint and Tomanos (2010).

Bibliography

Knowledge is of two kinds.
We know a subject ourselves, or we know
where we can find information upon it.

Samuel Johnson

M. Abramowitz and I. Stegun. *Handbook of Mathematical Functions*, vol. 55 of *National Bureau of Standards Applied Mathematics Series*. U.S. Government Printing Office, 1964.

R. K. Ahuja, M. Kodialam, A. K. Mishra, and J. B. Orlin. Computational investigation of maximum flow algorithms. *Eur. J. Oper. Res.*, **97**, 509–542, 1997.

R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows, Theory, Algorithms, and Applications*. Prentice-Hall, Englewood Cliffs, NJ, USA, 1993.

M. Al-Baali. Descent property and global convergence of the Fletcher-Reeves method with inexact line search. *IMA J. Numer. Anal.*, **5**, 121–124, 1985.

M. Anitescu and F. Potra. Formulating dynamic multi-rigid-body contact problems with friction as solvable linear complementarity problems. Reports on Computational Mathematics 93, Department of Mathematics, University of Iowa, Iowa City, USA, 1996.

G. B. Arfken. *Mathematical Methods for Physicists*. Academic Press, London, second edition, 1970.

L. Armijo. Minimization of functions having Lipschitz-continuous first partial derivatives. *Pacific J. Math*, **16**, 1–3, 1966.

B. M. Averick, R. G. Carter, J. J. Moré, and G.-L. Xue. The MINPACK-2 test problem collection. Technical Report Preprint MCS-P153-0692, Mathematics and Computer Science, Argonne National Laboratory, Argonne, IL, USA, June 1992.

R. E. Baker, S. Schnell, and P. K. Maini. Waves and patterning in developmental biology: vertebrate segmentation and feather bud formation as case studies. *Int. J. Dev. Biol.*, **53**, 783–794, 2009.

R. A. Barrio, C. Varea, J. L. Aragón, and P. K. Maini. A two-dimensional numerical study of spatial pattern formation in interacting Turing systems. *Bull. Math. Biol.*, **61**(3), 483–505, 1999.

R. H. Bartels and G. W. Stewart. A solution of the equation $AX + XB = C$. *ACM Trans. Math. Softw.*, **15**, 820–826, 1972.

F. Bastin, C. Cirillo, and Ph. L. Toint. An adaptive Monte-Carlo algorithm for computing mixed logit estimators. *Comput. Manag. Sci.*, **3**(1), 55–80, 2006*a*.

F. Bastin, C. Cirillo, and Ph. L. Toint. Application of an adaptive Monte-Carlo algorithm to mixed logit estimation. *Transport. Res. B*, **40**(7), 577–593, 2006*b*.

F. Bastin, V. Malmedy, M. Mouffe, Ph. L. Toint, and D. Tomanos. A retrospective trust-region method for unconstrained optimization. *Math. Program. Ser. A*, **123**(2), 395–418, 2010.

H. G. Bock and K. J. Plitt. A multiple shooting algorithm for direct solution of optimal control problems. In *Proc. 9th IFAC World Congress Budapest*. Pergamon Press, 1984.

J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications*. North Holland, 1976.

A. Brandt. Multi-level adaptative technique (mlat) for fast numerical solution to boundary value problems. In *Proc. 3rd Int. Conf. on Numerical Methods in Fluid Mechanics*, vol. 1, pp. 82–89, 1973.

W. L. Briggs, V. E. Henson, and S. F. McCormick. *A Multigrid Tutorial*. SIAM, Philadelphia, USA, second edition, 2000.

M. M. Bronstein, A. M. Bronstein, R. Kimmel, and I. Yavneh. A multigrid approach for multi-dimensional scaling. Talk at the 12th Copper Mountain Conference on Multigrid Methods, 2005.

C. G. Broyden. A class of methods for solving nonlinear simultaneous equations. *Math. Comput.*, **19**, 577–593, 1965.

C. G. Broyden. Quasi-newton methods and their application to function minimization. *Math. Comput.*, **21**, 368–381, 1967.

C. G. Broyden. The convergence of a class of double-rank minimization algorithms. *J. Inst. Math. Appl.*, **6**, 76–90, 1970.

A. Buckley and A. LeNir. QN-like variable storage conjugate gradients. *Math. Program.*, **27**(2), 155–175, 1983.

A. G. Buckley. A combined conjugate-gradient quasi-Newton minimization algorithm. *Math. Program.*, **15**, 200–210, 1978*a*.

A. G. Buckley. Extending the relationship between the conjugate gradient and the BFGS algorithms. *Math. Program.*, **15**, 343–348, 1978*b*.

R. H. Byrd, J. Nocedal, and R. B. Schnabel. Representation of quasi-newton matrices and their use in limited meory methods. *Math. Program.*, **63**, 129–156, 1994.

T. Carletti, J.-C. Delvenne, A. Füzfa, D. Lambert, V. Malmedy, and Ph. L. Toint. Characterization of complexity in reaction-diffusion patterns. 2010. (in preparation).

A. Cauchy. Méthode générale pour la résolution des systèmes d'équations simultanées. *Comptes Rendus de l'Académie des Sciences*, pp. 536–538, 1847.

C. Chang, P. Wu, R. E. Baker, P. K. Maini, L. Alibardi, and C.-M. Chuong. Reptile scale paradigm: Evo-devo, pattern formation and regeneration. *Int. J. Dev. Biol.*, **53**, 813–826, 2009.

B. V. Cherkassky. A fast algorithm for computing maximum flow in a network. In A. V. Karzanov, ed., *Collected Papers, vol. 3: Combinatorial Methods for Flow Problems*, Moscow, 1979. The Institute for Systems Studies. In Russian. English translation appears in AMS Trans., vol. 158, pp. 23–30, 1994.

B. V. Cherkassky and A. V. Goldberg. On implementing the push-relabel method for the maximum flow problem. *Algorithmica*, **19**(4), 390–410, 1997.

T. F. Coleman and J. J. Moré. Estimation of sparse Hessian matrices and graph coloring problems. *Math. Program.*, **28**, 243–270, 1984.

T. F. Coleman, B. Garbow, and J. J. Moré. Software for estimating sparse Hessian matrices. *ACM Trans. Math. Softw.*, **11**, 363–378, 1985.

A. R. Conn, N. I. M. Gould, and Ph. L. Toint. Global convergence of a class of trust region algorithms for optimization with simple bounds. *SIAM J. Numer. Anal.*, **25**(182), 433–460, 1988*a*. See also same journal **26**, 764–767, 1989.

A. R. Conn, N. I. M. Gould, and Ph. L. Toint. Testing a class of methods for solving minimization problems with simple bounds on the variables. *Math. Comput.*, **50**, 399–430, 1988*b*.

A. R. Conn, N. I. M. Gould, and Ph. L. Toint. Convergence of quasi-Newton matrices generated by the symmetric rank one update. *Math. Program.*, **50**(2), 177–196, 1991.

A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *LANCELOT: a Fortran package for large-scale nonlinear optimization (Release A)*. Number 17 *in* 'Springer Series in Computational Mathematics'. Springer Verlag, Heidelberg, Berlin, New York, 1992.

A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *Trust-Region Methods*. Number 01 *in* 'MPS-SIAM Series on Optimization'. SIAM, Philadelphia, USA, 2000.

A. R. Conn, N. I. M. Gould, A. Sartenaer, and Ph. L. Toint. Global convergence of a class of trust region algorithms for optimization using inexact projections on convex constraints. *SIAM J. Optim.*, **3**(1), 164–221, 1993.

G. Cramer. *Introduction à l'analyse des lignes courbes algébriques*. Frères Cramer & Philibert, 1750.

H. B. Curry. The method of steepest descent for nonlinear minimization problems. *Q. Appl. Math.*, **2**, 258–261, 1944.

A. Curtis, M. J. D. Powell, and J. Reid. On the estimation of sparse Jacobian matrices. *J. Inst. Math. Appl.*, **13**, 117–119, 1974.

Y. Dai and Y. Yuan. An efficient hybrid conjugate gradient method for unconstrained optimization. *Ann. Oper. Res.*, **103**, 33–47, March 2001.

Y. H. Dai and Y. Yuan. A Nonlinear Conjugate Gradient Method with a Strong Global Convergence Property. *SIAM J. Optim.*, **10**(1), 177–182, 1999.

W. C. Davidon. Variable metric method for minimization. Report ANL-5990(Rev.), Argonne National Laboratory, Research and Development, 1959. Republished in SIAM J. Optim., **1**, 1–17, 1991.

W. C. Davidon. Variance algorithms for minimization. *Computer J.*, **10**, 406–410, 1968.

T. A. Davis. *Direct Methods for Sparse Linear Systems*. Number 2 *in* 'Fundamentals of Algorithms'. SIAM, Philadelphia, USA, 2006.

J.-C. Delvenne, S. N. Yaliraki, and M. Barahona. Stability of graph communities across time scales. arXiv:0812.1811, 2009.

J. E. Dennis. A brief introduction to quasi-Newton methods. In G. H. Golub and J. Oliger, eds., *Numerical Analysis*, number 22 *in* 'Proceedings of Symposia in Applied Mathematics', pp. 19–52, Providence, RI, USA, 1978. American Mathematical Society.

J. E. Dennis and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice-Hall, Englewood Cliffs, NJ, USA, 1983. Reprinted as *Classics in Applied Mathematics 16*, SIAM, Philadelphia, USA, 1996.

J. E. Dennis, Jr. and J. J. Moré. A characterization of superlinear convergence and its application to quasi-Newton methods. *Math. Comput.*, **28**(126), 549–560, 1974.

J. E. Dennis, Jr. and J. J. Moré. Quasi-Newton methods, motivation and theory. *SIAM Rev.*, **19**, 46–89, 1977.

U. Derigs and W. Meier. Implementing goldberg's max-flow algorithm — a computational investigation. *Z. Oper. Res.-Meth. Model. Oper. Res.*, **33**, 383–403, 1989.

E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Math. Program.*, **91**(2), 201–213, 2002.

A. Ern and J.-L. Guermond. *Theory and Practice of Finite Elements.* Springer Verlag, 2004.

R. P. Fedorenko. The speed of convergence of one iterative process. *USSR Comput. Math. and Math. Phys.*, **4**(3), 227–235, 1964.

P. Fermat. Letter to Cureau de la Chambre. January 1662.

A. V. Fiacco and G. P. McCormick. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques.* J. Wiley and Sons, Chichester, England, 1968. Reprinted as *Classics in Applied Mathematics 4*, SIAM, 1990.

M. Fisher. Minimization algorithms for variational data assimilation. In *Recent Developments in Numerical Methods for Atmospheric Modelling*, pp. 364–385. ECMWF, 1998.

R. Fletcher. A new approach to variable metric algorithms. *Computer J.*, **13**, 317–322, 1970.

R. Fletcher. *Practical Methods of Optimization.* J. Wiley and Sons, Chichester, England, second edition, 1987.

R. Fletcher and S. Leyffer. Nonlinear programming without a penalty function. *Math. Program.*, **91**(2), 239–269, 2002.

R. Fletcher and M. J. D. Powell. A rapidly convergent descent method for minimization. *Computer J.*, **6**, 163–168, 1963.

R. Fletcher and C. M. Reeves. Function minimization by conjugate gradients. *Computer J.*, **7**, 149–154, 1964.

R. Fletcher, S. Leyffer, and Ph. L. Toint. On the global convergence of a filter-SQP algorithm. *SIAM J. Optim.*, **13**(1), 44–59, 2002.

M. Garland. *Quadric-Based Polygonal Surface Simplification.* PhD thesis, University of Illinois, Urbana-Champaign, 1999.

A. H. Gebremedhin, F. Manne, and A. Pothen. What color is your Jacobian? graph coloring for computing derivatives. *SIAM Rev.*, **47**(4), 629–705, 2005.

J. C. Gilbert and C. Lemaréchal. Some numerical experiments with variable-storage quasi-Newton algorithms. *Math. Program. Ser. B*, **45**(1), 407–436, 1989.

P. E. Gill and W. Murray. Conjugate-gradient methods for large-scale nonlinear optimization. Technical Report SOL 79-15, Operations Research Department, Stanford University, Stanford, USA, 1979.

P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization*. Academic Press, London, 1981.

D. M. Glover, W. J. Jenkins, and S. C. Doney. Modeling methods for marine science. Lecture notes, Woods Hole Oceanographic Institution, Woods Hole, MA, 2005.

M. S. Gockenbach. *Understanding and Implementing the Finite Element Method*. SIAM, Philadelphia, USA, 2006.

A. R. Goldberg and R. E. Tarjan. A new approach to the maximum-flow problem. *J. ACM*, **35**(4), 921–940, October 1988.

D. Goldfarb. A family of variable metric methods derived by variational means. *Math. Comput.*, **24**, 23–26, 1970.

D. Goldfarb. Factorized variable metric methods for unconstrained optimization. *Math. Comput.*, **30**(136), 796–811, 1976.

D. Goldfarb and Ph. L. Toint. Optimal estimation of Jacobian and Hessian matrices that arise in finite difference calculations. *Math. Comput.*, **43**(167), 69–88, 1984.

S. M. Goldfeldt, R. E. Quandt, and H. F. Trotter. Maximization by quadratic hill-climbing. *Econometrica*, **34**, 541–551, 1966.

A. A. Goldstein. *Constructive real analysis*. Harper's series in modern mathematics. Harper & Row, New York, 1967.

G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, third edition, 1996.

N. I. M. Gould, D. Orban, and Ph. L. Toint. CUTEr, a constrained and unconstrained testing environment, revisited. *ACM Trans. Math. Softw.*, **29**(4), 373–394, 2003*a*.

N. I. M. Gould, D. Orban, and Ph. L. Toint. GALAHAD—a library of thread-safe Fortran 90 packages for large-scale nonlinear optimization. *ACM Trans. Math. Softw.*, **29**(4), 353–372, 2003*b*.

S. Gratton and Ph. L. Toint. Approximate invariant subspaces and quasi-newton optimization methods. *Optim. Method Softw.*, **25**(4), 507–529, 2010.

S. Gratton, V. Malmedy, and Ph. L. Toint. Using approximate secant equations in limited memory methods for multilevel unconstrained optimization. *Comput. Optim. Appl.*, 2010*a*. (to appear).

S. Gratton, M. Mouffe, A. Sartenaer, Ph. L. Toint, and D. Tomanos. Numerical experience with a recursive trust-region method for multilevel nonlinear optimization. *Optim. Method Softw.*, **25**(3), 359–386, 2010*b*.

S. Gratton, M. Mouffe, Ph. L. Toint, and M. Weber-Mendonça. A recursive trust-region method in infinity norm for bound-constrained nonlinear optimization. *IMA J. Numer. Anal.*, **28**(4), 827–861, 2008*a*.

S. Gratton, A. Sartenaer, and Ph. L. Toint. Recursive Trust-Region Methods for Multiscale Nonlinear Optimization. *SIAM J. Optim.*, **19**(1), 414–444, 2008*b*.

A. Griewank. On automatic differentiation. In M. Iri and K. Tanabe, eds., *Mathematical Programming: Recent Developments and Applications*, pp. 83–108, Dordrecht, The Netherlands, 1989. Kluwer Academic Publishers.

A. Griewank. The global convergence of partitioned BFGS on problems with convex decompositions and Lipschitzian gradients. *Math. Program. Ser. A*, **50**(2), 1991.

A. Griewank and Ph. L. Toint. Local convergence analysis for partitioned quasi-Newton updates. *Numer. Math.*, **39**, 429–448, 1982*a*.

A. Griewank and Ph. L. Toint. On the unconstrained optimization of partially separable functions. In M. J. D. Powell, ed., *Nonlinear Optimization 1981*, pp. 301–312, London, 1982*b*. Academic Press.

A. Griewank and Ph. L. Toint. Partitioned variable metric updates for large structured optimization problems. *Numer. Math.*, **39**, 119–137, 1982*c*.

A. Griewank and Ph. L. Toint. Numerical experiments with partially separable optimization problems. In D. F. Griffiths, ed., *Numerical Analysis: Proceedings Dundee 1983*, number 1066 *in* 'Lecture Notes in Mathematics', pp. 203–220, Heidelberg, Berlin, New York, 1984*a*. Springer Verlag.

A. Griewank and Ph. L. Toint. On the existence of convex decomposition of partially separable functions. *Math. Program.*, **28**, 25–49, 1984*b*.

A. Griewank and A. Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Number 105 *in* 'Other Titles in Applied Mathematics'. SIAM, 2 edition, 2008.

R. E. Griffith and R. A. Stewart. A nonlinear programming technique for the optimization of continuous processing systems. *Manage. Sci.*, **7**, 379–392, 1961.

L. Grippo, F. Lampariello, and S. Lucidi. A nonmonotone line search technique for Newton's method. *SIAM J. Numer. Anal.*, **23**(4), 707–716, 1986.

L. Grippo, F. Lampariello, and S. Lucidi. A truncated Newton method with nonmonotone line search for unconstrained optimization. *J. Optim. Theory Appl.*, **60**(3), 401–419, 1989.

L. Grippo, F. Lampariello, and S. Lucidi. A class of nonmonotone stabilization methods in unconstrained optimization. *Numer. Math.*, **59**, 779–805, 1991.

W. Hackbusch. Ein iteratives Verfharen zur schnellen Auflösung elliptischer Randwertprobleme. Report 76-12, Mathematisches Institut, Universität zu Köln, 1976.

W. Hackbusch. *Multi-grid Methods and Applications.* Number 4 *in* 'Springer Series in Computational Mathematics'. Springer Verlag, Heidelberg, Berlin, New York, 1995.

W. W. Hager and H. Zhang. A new conjugate gradient method with guaranteed descent and an efficient line search. *SIAM J. Optim.*, **16**(1), 170–192, 2005.

W. W. Hager and H. Zhang. Algorithm 851: CG_DESCENT, a conjugate gradient method with guaranteed descent. *ACM Trans. Math. Softw.*, **32**(1), 113–137, 2006*a*.

W. W. Hager and H. Zhang. A survey of nonlinear conjugate gradient methods. *Pacific J. Optim.*, **2**, 35–58, 2006*b*.

T. C. Hales. The honeycomb conjecture. *Discret. Comput. Geom.*, **25**(1), 1–22, 2001.

P. S. Heckbert and M. Garland. Optimal triangulation and quadric-based surface simplification. *Comput. Geom. Theory Appl.*, **14**(1-3), 49–65, 1999.

M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *J. Res. Natl. Bur. Stand.*, **49**(6), 409–436, 1952.

S. Hildebrandt and A. Tromba. *Mathématiques et formes optimales : L'explication des structures naturelles.* L'univers des sciences. Pour la Science, Belin, 1998.

R. A. Horn and C. R. Johnson. *Matrix Analysis.* Cambridge University Press, 1985.

S. Hossain and T. Steihaug. Computing a sparse Jacobian matrix by rows and columns. *Optim. Method Softw.*, **10**(1), 33–48, 1998.

S. Hossain and T. Steihaug. Sparsity issues in the computation of Jacobian matrices. In *ISSAC '02: Proceedings of the 2002 international symposium on Symbolic and algebraic computation*, pp. 123–130, New York, NY, USA, 2002. ACM.

John of Salisbury. *Metalogicon.* 1159.

E. Kalnay. *Atmospheric Modeling, Data Assimilation and Predictability.* Cambridge University Press, 2002.

T. Lalesco. *La géométrie du triangle.* Éditions Jacques Gabay, Paris, 2 edition, 1952.

C. Lánczos. *The Variational Principles of Mechanics.* Courier Dover Publications, 1986.

K. Levenberg. A method for the solution of certain problems in least squares. *Q. Appl. Math.*, **2**, 164–168, 1944.

J. M. Lewis, S. Lakshmivarahan, and S. Dhall. *Dynamic Data Assimilation: A Least Squares Approach*, vol. 104 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 2006.

M. Lewis and S. G. Nash. Model problems for the multigrid optimization of systems governed by differential equations. *SIAM J. Sci. Comput.*, **26**(6), 1811–1837, 2005.

C.-M. Lin, T. X. Jiang, R. E. Baker, P. K. Maini, R. B. Widelitz, and C.-M. Chuong. Spots and stripes: Pleomorphic patterning of stem cells via p-erk-dependent cell chemotaxis shown by feather morphogenesis and mathematical simulation. *Dev. Biol.*, **334**(2), 369–382, 2009.

D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Math. Program. Ser. B*, **45**(1), 503–528, 1989.

J. Loos, G. Greiner, and H.-P. Seidel. Computer aided spectacle lens design. Technical Report 5, Department of Computer Science, University of Erlangen, Erlangen, Germany, 1997.

L. Lovász. Random walks on graphs: A survey. In D. Miklos, V. T. Sos and T. Szonyi, eds., *Combinatorics, Paul Erdos is Eighty*, vol. 2, pp. 1–46, Keszthely, Hungary, 1993. Bolyai Society Mathematical Studies.

D. G. Luenberger. *Optimization by Vector Space Methods*. J. Wiley and Sons, Chichester, England, 1969.

D. G. Luenberger. *Information Science*. Princeton University Press, 2006.

C. Maclaurin. *A treatise of Algebra*. Millar & Nourse, London, 1748.

P. K. Maini, M. R. Myerscough, K. H. Winter, and J. D. Murray. Bifurcating spatially heterogeneous solutions in a chemotaxis model for biological pattern generation. *Bull. Math. Biol.*, **53**(5), 701–719, 1991.

V. Malmedy and Ph. L. Toint. Approximating hessians in unconstrained optimization arising from discretized problems. *Comput. Optim. Appl.*, 2010. (to appear).

V. Malmedy, Ph. L. Toint, and D. Tomanos. RMTR, a fortran package for multilevel nonlinear bound-constrained optimization. 2010. (in preparation).

O. L. Mangasarian. *Nonlinear Programming*. McGraw-Hill, New York, NY, USA, 1979.

D. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *SIAM J. Appl. Math.*, **11**, 431–441, 1963.

E. Marwil. *Exploiting sparsity in Newton-like methods*. PhD thesis, Cornell University, Ithaca, USA, 1978.

P. L. M. Maupertuis. Les loix du mouvement et du repos, déduites d'un principe de métaphysique. In *Histoire de l'Académie Royale des Sciences et des Belles-Lettres*, pp. 267–294, Berlin, 1746.

G. Meurant. *The Lanczos and Conjugate-Gradient Algorithms.* SIAM, Philadelphia, USA, 2006.

J. J. Moré and D. C. Sorensen. Computing a trust region step. *SIAM J. Sci. Stat. Comp.*, **4**(3), 553–572, 1983.

J. J. Moré and J. D. Thuente. Line search algorithms with guaranteed sufficient decrease. *ACM Trans. Math. Softw.*, **20**, 286–307, 1994.

D. D. Morrison. Methods for nonlinear least squares problems and convergence proofs. In J. Lorell and F. Yagi, eds., *Proceedings of the Seminar on Tracking Programs and Orbit Determination*, pp. 1–9, Pasadena, USA, 1960. Jet Propulsion Laboratory.

M. Mouffe. Etude des méthodes multigrilles dans le cadre de la résolution de systèmes linéaires. Master thesis, Department of Mathematics, University of Namur, Namur, Belgium, June 2005.

M. Mouffe. *Multilevel optimization in infinity norm and associated stopping criteria.* PhD thesis, Department of Mathematics, University of Namur, Namur, Belgium, February 2009.

J. D. Murray. *Mathematical Biology: Spatial models and biomedical applications*, vol. 2 of *Mathematical Biology.* Springer, third edition, 2003.

B. A. Murtagh and R. W. H. Sargent. A constrained minimization method with quadratic convergence. In R. Fletcher, ed., *Optimization*, pp. 215–246, London, 1969. Academic Press.

M. Nakahara. *Geometry, Topology and Physics.* CRC Press, second edition, 2003.

S. G. Nash. A multigrid approach to discretized optimization problems. *Optim. Method Softw.*, **14**, 99–116, 2000*a*.

S. G. Nash. A survey of truncated-Newton methods. *Journal of Computational and Applied Mathematics*, **124**, 45–59, 2000*b*.

L. Nazareth. A relationship between the BFGS and conjugate gradient algorithms and its implications for new algorithms. *SIAM J. Numer. Anal.*, **16**, 794–800, 1979.

J. Nocedal. Updating quasi-Newton matrices with limited storage. *Math. Comput.*, **35**, 773–782, 1980.

J. Nocedal and S. J. Wright. *Numerical Optimization.* Series in Operations Research. Springer Verlag, Heidelberg, Berlin, New York, second edition, 2006.

S. Oh, A. Milstein, C. Bouman, and K. Webb. Multigrid algorithms for optimization and inverse problems. In C. Bouman and R. Stevenson, eds., *Computational Imaging*, vol. 5016 of *Proceedings of the SPIE*, pp. 59–70. DDM, 2003.

J. M. Ortega and W. C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables*. Academic Press, London, 1970.

G. F. Oster and J. D. Murray. Pattern formation models and developmental constraints. *Journal of Experimental Zoology*, **251**(2), 186–202, 1989.

K. J. Painter, P. K. Maini, and H. G. Othmer. Stripe formation in juvenile pomacanthus explained by a generalized turing mechanism with chemotaxis. *Proc. Natl. Acad. Sci. USA*, **96**, 5549–5554, 1999.

J. M. Perry. A class of conjugate gradient algorithms with a two-step variable-metric memory. Discussion Paper 269, Center for Mathematical Studies in Economics and Management Science, Northwestern University, Evanston, IL, 1977.

R. Plaza, F. Sánchez-Garduño, P. Padilla, R. Barrio, and P. Maini. The effect of growth and curvature on pattern formation. *J. Dyn. Differ. Equ.*, **16**(4), 1093–1121, 2004.

K. J. Plitt. Ein superlinear konvergentes Mehrzielverfahren zur direkten Berechnung beschränkter optimaler steuerungen. Master's thesis, University of Bonn, 1981.

E. Polak and G. Ribière. Note sur la convergence de méthodes de directions conjuguées. *Revue française d'informatique et de recherche opérationelle*, **16-R1**, 35–43, 1969.

B. T. Polyak. The conjugate gradient method in extremal problems. *USSR Comput. Math. Math. Phys.*, **9**, 94–112, 1969.

M. J. D. Powell. A new algorithm for unconstrained optimization. In J. B. Rosen, O. L. Mangasarian and K. Ritter, eds., *Nonlinear Programming*, pp. 31–65, London, 1970. Academic Press.

M. J. D. Powell. Nonconvex minimization calculations and the conjugate gradient method. In D. F. Griffiths, ed., *Numerical Analysis, Proceedings Dundee 1983*, pp. 122–141, Heidelberg, Berlin, New York, 1984. Springer Verlag. Lecture Notes in Mathematics 1066.

M. J. D. Powell and Ph. L. Toint. On the estimation of sparse Hessian matrices. *SIAM J. Numer. Anal.*, **16**(6), 1060–1074, 1979.

M. J. D. Powell and Ph. L. Toint. The Shanno-Toint procedure for updating sparse symmetric matrices. *IMA J. Numer. Anal.*, **1**, 403–413, 1981.

R. Pytlak. *Conjugate Gradient Algorithms in Nonconvex Optimization*. Springer Verlag, Heidelberg, Berlin, New York, 2009.

S. M. Robinson. Analysis of sample-path optimization. *Mathematics of Operations Research*, **21**(3), 513–528, 1996.

R. T. Rockafellar. *Convex Analyis*. Princeton University Press, Princeton, USA, 1970.

Y. Saad. SPARSEKIT: a basic tool kit for sparse matrix computations. Technical Report RIACS-90-20, Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffett Field, CA, USA, June 1994. Version 2.

Y. Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, Philadelphia, USA, second edition, 2003.

E. W. Sachs. PDE constrained optimization. *SIAG/OPT News-and Views*, **14**(1), 7–10, 2003.

A. Schmidt and K. G. Siebert. *Design of Adaptive Finite Element Software: the finite element toolbox ALBERTA*. Springer Verlag, 2005.

R. B. Schnabel. Quasi-newton methods using multiple secant equations. Technical Report CU-CS-247-83, University of Colorado at Boulder, Departement of Computer Science, June 1983.

D. F. Shanno. Conditioning of quasi-Newton methods for function minimization. *Math. Comput.*, **24**, 647–657, 1970.

D. F. Shanno. Conjugate gradient methods with inexact searches. *Math. Oper. Res.*, **3**, 244–256, 1978*a*.

D. F. Shanno. On the convergence of a new conjugate gradient algorithm. *SIAM J. Numer. Anal.*, **15**(6), 1247–1257, 1978*b*.

A. Shapiro. Monte Carlo sampling methods. In A. Shapiro and A. Ruszczyński, eds., *Stochastic Programming*, vol. 10 of *Handbooks in Operations Research and Management Science*, pp. 353–425. Elsevier, Amsterdam, The Netherlands, 2003.

D. C. Sorensen. An example concerning quasi-Newton estimates of a sparse Hessian. *SIGNUM Newsletter*, **16**(2), 8–10, 1981.

T. Steihaug. The conjugate gradient method and trust regions in large scale optimization. *SIAM J. Numer. Anal.*, **20**(3), 626–637, 1983.

G. W. Stewart. A modification of Davidon's minimization method to accept difference approximations of derivativesc. *J. ACM*, **14**, 72–83, 1967.

J. C. Strikwerda. *Finite Difference Schemes and Partial Differential Equations*. SIAM, 2004.

Ph. L. Toint. On sparse and symmetric matrix updating subject to a linear equation. *Math. Comput.*, **31**(140), 954–961, 1977.

Ph. L. Toint. On the superlinear convergence of an algorithm for solving a sparse minimization problem. *SIAM J. Numer. Anal.*, **16**, 1036–1045, 1979.

Ph. L. Toint. A note on sparsity exploiting quasi-Newton methods. *Math. Program.*, **21**(2), 172–181, 1981*a*.

Ph. L. Toint. Towards an efficient sparsity exploiting Newton method for minimization. In I. S. Duff, ed., *Sparse Matrices and Their Uses*, pp. 57–88, London, 1981*b*. Academic Press.

Ph. L. Toint. Test problems for partially separable optimization and results for the routine PSPMIN. Technical Report 83/4, Department of Mathematics, University of Namur, Namur, Belgium, 1983.

Ph. L. Toint and D. Tomanos. Optimisation numérique appliquée, le design des lentilles progressives. *Tangente*, **126**, 48–50, 2009.

D. Tomanos. *Algorithms and Software for Multilevel Nonlinear Optimization*. PhD thesis, University of Namur (FUNDP), Namur, 2009.

U. Trottenberg, C. W. Oosterlee, and A. Schüller. *Multigrid*. Elsevier, Amsterdam, The Netherlands, 2001.

A. M. Turing. The chemical basis of morphogenesis. *Phil. Trans. R. Soc. Lond. B*, **237**, 37–72, 1952.

M. Weber Mendonça. *Multilevel Optimization: Convergence Theory, Algorithms and Application to Derivative-Free Optimization*. PhD thesis, Department of Mathematics, University of Namur, Namur, Belgium, 2009.

Z. Wen and D. Goldfarb. A linesearch multigrid methods for large-scale convex optimization. Technical report, Department of Industrial Engineering and Operations Research, Columbia University, New York, July 2007.

P. Wesseling. *An introduction to Multigrid Methods*. J. Wiley and Sons, Chichester, England, 1992. Corrected Reprint, Edwards, Philadelphia, 2004.

K. H. Winters, M. R. Myerscough, P. K. Maini, and J. D. Murray. Tracking bifurcating solutions of a model biological pattern generator. *IMPACT Comput. Sci. Eng.*, **2**(4), 355–371, 1990.

P. Wolfe. Another variable metric method. Working paper, 1968.

P. Wolfe. Convergence conditions for ascent methods. *SIAM Rev.*, **11**, 226–235, 1969.

P. Wolfe. Convergence conditions for ascent methods. ii: Some corrections. *SIAM Rev.*, **13**, 185–188, 1971.

H. C. Zhang and W. W. Hager. A nonmonotone line search technique and its application to unconstrained optimization. *SIAM J. Optim.*, **14**, 1043–1056, 2004.

J. Zhu, Y.-T. Zhang, S. A. Newman, and M. Alber. Application of discontinuous galerkin methods for reaction-diffusion systems in developmental biology. *J. Sci. Comput.*, **40**, 391–418, 2009.

J. C. Ziems and S. Ulbrich. Adaptive multilevel inexact SQP-methods for PDE-constrained optimization. Technical report, TU Darmstadt, 2008.

Main notations and abbreviations

General

$\mathbb{N}$	set of positive integers	9
$\mathbb{R}$	set of real numbers	3
$\mathbb{R}^+$	set of nonnegative real numbers	4
$\mathbb{R}^n$	real n-dimensional Euclidean space	3
$\mathbb{R}^{m\times n}$	set of $m \times n$ real matrices	5
$\lvert\mathcal{S}\rvert$	cardinality of set $\mathcal{S}$	3
$\lVert x\rVert$	norm of vector x (Euclidean unless otherwise specified)	4
$\lVert A\rVert$	norm of matrix A (2-norm unless otherwise specified)	7
$\langle x, y\rangle$	inner product of vectors x and y	4
$A \bullet B$	Hadamard product of matrices A and B	6
$A^{\maltese}$	inverse of matrix A for the Hadamard product	6
$A \otimes B$	Kronecker product of matrix A by matrix B	6
$\operatorname{vec} A$	vectorized form of matrix A	6
$\rho(A)$	spectral radius of matrix A	7
$\kappa(A)$	condition number of matrix A	7
$\sigma_{\min}(A)$	smallest singular value of matrix A	7
$\sigma_{\max}(A)$	largest singular value of matrix A	7
$\partial\mathcal{S}$	boundary of set $\mathcal{S}$	8
$[x, y]$	segment between point x and y	8
$\nabla f(x)$	gradient of function f at point x	11
$\nabla^2 f(x)$	Hessian of function f at point x	12
Δ	Laplace-Beltrami operator	13

Fundamentals of nonlinear optimization

α_k	step length at iteration k	27
B_k	Hessian approximation at iteration k	25
$\mathcal{B}_k$	trust region at iteration k	40
d_k	search direction at iteration k	27
Δ_k	trust-region radius at iteration k	40
f_k	value of the objective function f at x_k	24
g_k	gradient of the objective function f at x_k	24
H_k	inverse Hessian approximation at iteration k	84
m_k	model of function f at iteration k	40
ρ_k	ratio of achieved reduction to predicted reduction	40
s_k	step at iteration k	24
x_*	optimal solution	9
x_k	k-th iterate	8
y_k	variation of the gradient along the step s_k	38

Constants

ϵ_{M}	machine precision	82
κ_{lbd}	**l**ower **b**ound on **d**elta (Δ)	49
κ_{lbg}	**l**ower **b**ound on **g**radient	49
κ_{lch}	**L**ipschitz **c**onstant for the model's **H**essian	50
κ_{mdc}	fraction of **m**odel **dec**rease in the Cauchy condition	45
κ_{mqd}	**m**odel **q**uadratic **d**ecrease	51
κ_{sod}	**s**econd-**o**rder **d**ecrease	50
κ_{sr1}	threshold for collinearity control in SR1 update	85
κ_{ubh}	**u**pper **b**ound on the norm of the function and model **H**essians	46
κ_{ufh}	**u**pper bound on the norm of the **f**unction **H**essian	45
κ_{umh}	**u**pper bound on the norm of the **m**odel **H**essian	45

Multilevel framework

f_i	objective function at level i	69
h_i, $h_{i,k}$	coarse model at level i (and at iteration k)	71
i	level counter	69
k	iteration counter	71
m_i, $m_{i,k}$	Taylor's model at level i (and at iteration k)	71
P_i	prolongation operator from level $i-1$ to level i	70
R_i	restriction operator from level i to level $i-1$	70

Partial separability and sparsity

B_i	approximate Hessian of the i-th element function	95
f_i	i-th element function	10
g_i	gradient of the i-th element function	95
I_i	identity matrix with zeros at diagonal entries indexed by $\mathcal{I}_i^{\mathsf{c}}$	95
$\mathcal{I}_i$	set of indices of the variables on which f_i depends	95
$\mathcal{I}_i^{\mathsf{c}}$	complementary set of $\mathcal{I}_i$	95
J_i	matrix with zero entries everywhere except in positions $\mathcal{I}_i \times \mathcal{I}_i$ where they are equal to 1	95
$\mathcal{P}$	gangster operator (enforcing some sparsity pattern)	94
$\mathcal{S}$	sparsity pattern	89
s_i	step vector s whose components in $\mathcal{I}_i^{\mathsf{c}}$ have been zeroed	95

Abbreviations

AF	All on Finest	76
BFGS	Broyden-Fletcher-Goldfarb-Shanno	86
BP-BFGS	Balanced Partitioned BFGS	105
BTR	Basic Trust-Region	40
CG	Conjugate gradient	34
DFP	Davidon-Fletcher-Powell	85
DS	Dennis-Schnabel	28
DY	Dai-Yuan	39
DYHS	Dai-Yuan-Hestenes-Stiefel	39
FM	Full Multilevel	76
FR	Fletcher-Reeves	38
HS	Hestenes-Stiefel	38
HZ	Hager-Zhang	39
L-BFGS	Limited-memory BFGS	115
LTS	Lower-Triangular Substitution	91
MF	Multilevel on Finest	76
MR	Mesh Refinement	76
MS	Moré-Sorensen	55
NCG	Nonlinear conjugate gradient	39
ODE	Ordinary differential equations	13
PCG	Preconditioned Conjugate gradient	36
PDE	Partial differential equations	12
PRP	Polak-Ribière-Polyak	39

PS-PSB	Partially Separable PSB	98
PSB	Powell-symmetric-Broyden	85
PTCG	Projected Truncated Conjugate Gradient	76
RMTR	Recursive Multilevel Trust-Region	59
RTR	Retrospective Trust-Region	43
S-PSB	Sparse PSB	95
SCM	Sequential Coordinate Minimization	77
SR1	Symmetric-Rank-One	85
TCG	Truncated Conjugate gradient	37
UP-BFGS	Uniformly Partitioned BFGS	101

List of Algorithms

List of Figures

List of Tables

Index

Appendices

Appendix A

Detailed numerical results

A.1 Retrospective trust-region method

Here is the set of results from our tests. For each problem, we report its number of variables (n), whether it is a least-squares problem (marked by a star in column LS), the number of iterations (iter), the number of gradient evaluations (#g) and the best objective function value found (f). The symbol "–" indicates that the iteration limit (fixed at 50,000) was exceeded.

Name	LS	n	Moré-Sorensen						Steihaug-Toint					
			BTR			RTR			BTR			RTR		
			iter	#g	f	iter	#g	f	iter	#g	f	iter	#g	f
AKIVA		2	6	7	$6.166 \cdot 10^{+00}$	6	7	$6.166 \cdot 10^{+00}$	8	9	$6.166 \cdot 10^{+00}$	8	9	$6.166 \cdot 10^{+00}$
ALLINITU		4	7	8	$5.744 \cdot 10^{+00}$	7	8	$5.744 \cdot 10^{+00}$	5	6	$5.744 \cdot 10^{+00}$	5	6	$5.744 \cdot 10^{+00}$
ARGLINA	*	200	5	6	$2.000 \cdot 10^{+02}$	5	6	$2.000 \cdot 10^{+02}$	5	6	$2.000 \cdot 10^{+02}$	5	6	$2.000 \cdot 10^{+02}$
ARWHEAD		100	5	6	$6.595 \cdot 10^{-14}$	5	6	$6.595 \cdot 10^{-14}$	5	6	$0.000 \cdot 10^{+00}$	5	6	$0.000 \cdot 10^{+00}$
BARD	*	3	9	9	$8.215 \cdot 10^{-03}$	9	9	$8.215 \cdot 10^{-03}$	13	13	$8.215 \cdot 10^{-03}$	13	13	$8.215 \cdot 10^{-03}$
BDQRTIC	*	100	10	11	$3.788 \cdot 10^{+02}$	10	11	$3.788 \cdot 10^{+02}$	13	14	$3.788 \cdot 10^{+02}$	13	14	$3.788 \cdot 10^{+02}$
BEALE	*	2	9	9	$1.923 \cdot 10^{-16}$	8	8	$4.581 \cdot 10^{-14}$	7	8	$7.319 \cdot 10^{-12}$	7	8	$7.319 \cdot 10^{-12}$
BIGGS6	*	6	6,094	4,585	$2.427 \cdot 10^{-01}$	6,021	4,685	$2.427 \cdot 10^{-01}$	149	135	$8.947 \cdot 10^{-09}$	149	138	$1.649 \cdot 10^{-07}$
BOX3	*	3	7	8	$1.519 \cdot 10^{-11}$	7	8	$1.519 \cdot 10^{-11}$	8	9	$2.384 \cdot 10^{-15}$	8	9	$2.384 \cdot 10^{-15}$
BRKMCC		2	2	3	$1.690 \cdot 10^{-01}$	2	3	$1.690 \cdot 10^{-01}$	3	4	$1.690 \cdot 10^{-01}$	3	4	$1.690 \cdot 10^{-01}$
BROWNAL	*	200	24	20	$5.320 \cdot 10^{-23}$	32	27	$1.268 \cdot 10^{-15}$	5	6	$1.473 \cdot 10^{-09}$	5	6	$1.473 \cdot 10^{-09}$
BROWNBS	*	2	29	29	$0.000 \cdot 10^{+00}$	29	29	$0.000 \cdot 10^{+00}$	51	52	$0.000 \cdot 10^{+00}$	55	56	$0.000 \cdot 10^{+00}$
BROWNDEN	*	4	10	11	$8.582 \cdot 10^{+04}$	10	11	$8.582 \cdot 10^{+04}$	11	12	$8.582 \cdot 10^{+04}$	11	12	$8.582 \cdot 10^{+04}$
BROYDN7D		100	24	21	$3.974 \cdot 10^{+01}$	23	21	$3.977 \cdot 10^{+01}$	35	31	$3.966 \cdot 10^{+01}$	31	27	$3.966 \cdot 10^{+01}$
BRYBND	*	100	17	13	$2.069 \cdot 10^{-28}$	12	12	$1.412 \cdot 10^{-23}$	11	12	$2.866 \cdot 10^{-17}$	11	12	$2.866 \cdot 10^{-17}$
CHAINWOO	*	100	53	44	$1.000 \cdot 10^{+00}$	50	45	$1.000 \cdot 10^{+00}$	300	228	$5.504 \cdot 10^{+01}$	162	141	$3.219 \cdot 10^{+01}$
CHNROSNB	*	50	57	48	$1.892 \cdot 10^{-13}$	54	50	$2.484 \cdot 10^{-21}$	78	61	$6.734 \cdot 10^{-14}$	64	59	$2.226 \cdot 10^{-15}$
CLIFF		2	27	28	$1.998 \cdot 10^{-01}$	27	28	$1.998 \cdot 10^{-01}$	30	31	$1.998 \cdot 10^{-01}$	30	31	$1.998 \cdot 10^{-01}$
COSINE		100	6	7	$-9.900 \cdot 10^{+01}$	6	7	$-9.900 \cdot 10^{+01}$	10	10	$-9.900 \cdot 10^{+01}$	10	10	$-9.900 \cdot 10^{+01}$
CRAGGLVY		202	15	16	$6.674 \cdot 10^{+01}$	15	16	$6.674 \cdot 10^{+01}$	16	17	$6.674 \cdot 10^{+01}$	16	17	$6.674 \cdot 10^{+01}$
CUBE	*	2	37	31	$9.305 \cdot 10^{-12}$	35	31	$1.921 \cdot 10^{-15}$	44	38	$1.230 \cdot 10^{-12}$	42	37	$1.756 \cdot 10^{-13}$
CURLY10	*	50	9	10	$-5.016 \cdot 10^{+03}$	9	10	$-5.016 \cdot 10^{+03}$	18	18	$-5.016 \cdot 10^{+03}$	18	18	$-5.016 \cdot 10^{+03}$
CURLY20	*	50	8	9	$-5.016 \cdot 10^{+03}$	8	9	$-5.016 \cdot 10^{+03}$	18	18	$-5.016 \cdot 10^{+03}$	18	18	$-5.016 \cdot 10^{+03}$
CURLY30	*	50	13	13	$-5.016 \cdot 10^{+03}$	13	13	$-5.016 \cdot 10^{+03}$	17	16	$-5.016 \cdot 10^{+03}$	20	19	$-5.016 \cdot 10^{+03}$
DECONVU	*	61	25	19	$1.929 \cdot 10^{-10}$	19	16	$1.725 \cdot 10^{-08}$	22	19	$3.904 \cdot 10^{-08}$	22	20	$3.997 \cdot 10^{-08}$
DENSCHNA		2	5	6	$2.214 \cdot 10^{-12}$	5	6	$2.214 \cdot 10^{-12}$	5	6	$1.200 \cdot 10^{-15}$	5	6	$1.200 \cdot 10^{-15}$
DENSCHNB	*	2	4	5	$3.385 \cdot 10^{-16}$	4	5	$3.385 \cdot 10^{-16}$	6	7	$7.995 \cdot 10^{-14}$	6	7	$7.995 \cdot 10^{-14}$
DENSCHNC	*	2	10	11	$2.178 \cdot 10^{-20}$	10	11	$2.178 \cdot 10^{-20}$	9	10	$1.842 \cdot 10^{-13}$	9	10	$1.842 \cdot 10^{-13}$
DENSCHND	*	3	37	33	$1.139 \cdot 10^{-08}$	38	34	$1.139 \cdot 10^{-08}$	30	31	$1.375 \cdot 10^{-08}$	30	31	$1.375 \cdot 10^{-08}$
DENSCHNE	*	3	9	10	$8.710 \cdot 10^{-19}$	9	10	$8.710 \cdot 10^{-19}$	16	16	$4.459 \cdot 10^{-19}$	15	16	$7.381 \cdot 10^{-13}$
DENSCHNF	*	2	6	7	$6.513 \cdot 10^{-22}$	6	7	$6.513 \cdot 10^{-22}$	6	7	$6.513 \cdot 10^{-22}$	6	7	$6.513 \cdot 10^{-22}$
DIXMAANA		150	7	8	$1.000 \cdot 10^{+00}$	7	8	$1.000 \cdot 10^{+00}$	9	10	$1.000 \cdot 10^{+00}$	9	10	$1.000 \cdot 10^{+00}$
DIXMAANB		150	11	11	$1.000 \cdot 10^{+00}$	11	11	$1.000 \cdot 10^{+00}$	9	10	$1.000 \cdot 10^{+00}$	9	10	$1.000 \cdot 10^{+00}$

continued on next page...

Name	LS	n	Moré-Sorensen						Steihaug-Toint					
			BTR			RTR			BTR			RTR		
			iter	#g	f	iter	#g	f	iter	#g	f	iter	#g	f
DIXMAANC		150	11	11	$1.000 \cdot 10^{+00}$	11	11	$1.000 \cdot 10^{+00}$	10	11	$1.000 \cdot 10^{+00}$	10	11	$1.000 \cdot 10^{+00}$
DIXMAAND		150	14	13	$1.000 \cdot 10^{+00}$	14	13	$1.000 \cdot 10^{+00}$	11	12	$1.000 \cdot 10^{+00}$	11	12	$1.000 \cdot 10^{+00}$
DIXMAANE		150	10	10	$1.000 \cdot 10^{+00}$	11	11	$1.000 \cdot 10^{+00}$	11	11	$1.000 \cdot 10^{+00}$	11	12	$1.000 \cdot 10^{+00}$
DIXMAANF		150	15	14	$1.000 \cdot 10^{+00}$	14	13	$1.000 \cdot 10^{+00}$	12	13	$1.000 \cdot 10^{+00}$	12	13	$1.000 \cdot 10^{+00}$
DIXMAANG		150	15	14	$1.000 \cdot 10^{+00}$	15	14	$1.000 \cdot 10^{+00}$	13	14	$1.000 \cdot 10^{+00}$	13	14	$1.000 \cdot 10^{+00}$
DIXMAANH		150	18	16	$1.000 \cdot 10^{+00}$	19	17	$1.000 \cdot 10^{+00}$	14	15	$1.000 \cdot 10^{+00}$	14	15	$1.000 \cdot 10^{+00}$
DIXMAANI		150	14	14	$1.000 \cdot 10^{+00}$	16	16	$1.000 \cdot 10^{+00}$	13	14	$1.000 \cdot 10^{+00}$	13	14	$1.000 \cdot 10^{+00}$
DIXMAANJ		150	25	21	$1.000 \cdot 10^{+00}$	18	16	$1.000 \cdot 10^{+00}$	18	17	$1.000 \cdot 10^{+00}$	19	18	$1.000 \cdot 10^{+00}$
DIXMAANK		150	23	20	$1.000 \cdot 10^{+00}$	19	17	$1.000 \cdot 10^{+00}$	22	20	$1.000 \cdot 10^{+00}$	20	19	$1.000 \cdot 10^{+00}$
DIXMAANL		150	23	20	$1.000 \cdot 10^{+00}$	25	22	$1.000 \cdot 10^{+00}$	15	16	$1.000 \cdot 10^{+00}$	15	16	$1.000 \cdot 10^{+00}$
DIXON3DQ		100	4	5	$1.171 \cdot 10^{-29}$	4	5	$1.171 \cdot 10^{-29}$	8	9	$0.000 \cdot 10^{+00}$	8	9	$0.000 \cdot 10^{+00}$
DJTL		2	105	71	$-8.952 \cdot 10^{+03}$	104	74	$-8.952 \cdot 10^{+03}$	231	161	$-8.952 \cdot 10^{+03}$	253	183	$-8.952 \cdot 10^{+03}$
DQDRTIC		100	5	6	$2.399 \cdot 10^{-28}$	5	6	$2.399 \cdot 10^{-28}$	9	10	$1.745 \cdot 10^{-17}$	9	10	$1.745 \cdot 10^{-17}$
DQRTIC		100	29	30	$2.806 \cdot 10^{-08}$	29	30	$2.806 \cdot 10^{-08}$	29	30	$3.590 \cdot 10^{-08}$	29	30	$3.590 \cdot 10^{-08}$
EDENSCH		100	19	18	$6.033 \cdot 10^{+02}$	20	19	$6.033 \cdot 10^{+02}$	17	18	$6.033 \cdot 10^{+02}$	17	18	$6.033 \cdot 10^{+02}$
EG2		100	3	4	$-9.895 \cdot 10^{+01}$	3	4	$-9.895 \cdot 10^{+01}$	3	4	$-9.895 \cdot 10^{+01}$	3	4	$-9.895 \cdot 10^{+01}$
EIGENALS	*	110	20	21	$5.077 \cdot 10^{-21}$	20	20	$1.111 \cdot 10^{-12}$	23	23	$1.053 \cdot 10^{-12}$	23	23	$8.333 \cdot 10^{-13}$
EIGENBLS	*	110	134	107	$4.241 \cdot 10^{-15}$	69	63	$3.185 \cdot 10^{-17}$	164	142	$3.794 \cdot 10^{-13}$	167	153	$1.343 \cdot 10^{-12}$
ENGVAL1		100	9	10	$1.091 \cdot 10^{+02}$	9	10	$1.091 \cdot 10^{+02}$	11	12	$1.091 \cdot 10^{+02}$	11	12	$1.091 \cdot 10^{+02}$
ENGVAL2	*	3	13	14	$9.715 \cdot 10^{-17}$	13	14	$9.715 \cdot 10^{-17}$	24	24	$5.201 \cdot 10^{-15}$	24	24	$1.195 \cdot 10^{-15}$
ERRINROS	*	50	56	48	$3.990 \cdot 10^{+01}$	52	47	$3.990 \cdot 10^{+01}$	85	79	$3.990 \cdot 10^{+01}$	75	72	$3.990 \cdot 10^{+01}$
EXPFIT	*	2	7	6	$2.405 \cdot 10^{-01}$	7	6	$2.405 \cdot 10^{-01}$	13	12	$2.405 \cdot 10^{-01}$	16	14	$2.405 \cdot 10^{-01}$
EXTROSNB	*	100	1,281	1,182	$1.837 \cdot 10^{-08}$	487	468	$3.172 \cdot 10^{-07}$	566	516	$1.578 \cdot 10^{-06}$	643	624	$7.153 \cdot 10^{-07}$
FLETCBV2		100	2	3	$-5.140 \cdot 10^{-01}$	2	3	$-5.140 \cdot 10^{-01}$	3	4	$-5.140 \cdot 10^{-01}$	3	4	$-5.140 \cdot 10^{-01}$
FLETCBV3		50	–	–	$-3.507 \cdot 10^{+02}$	–	–	$-3.392 \cdot 10^{+02}$	30,878	30,541	$-1.386 \cdot 10^{+03}$	–	–	$-1.029 \cdot 10^{+03}$
FLETCHBV		10	460	453	$-2.150 \cdot 10^{+06}$	1,203	1,151	$-2.020 \cdot 10^{+06}$	127	118	$-2.367 \cdot 10^{+06}$	257	257	$-2.111 \cdot 10^{+06}$
FLETCHCR		100	231	200	$1.710 \cdot 10^{-19}$	164	162	$2.643 \cdot 10^{-19}$	347	264	$1.205 \cdot 10^{-14}$	194	180	$7.811 \cdot 10^{-18}$
FMINSRF2		121	35	31	$1.000 \cdot 10^{+00}$	30	25	$1.000 \cdot 10^{+00}$	95	91	$1.000 \cdot 10^{+00}$	70	60	$1.000 \cdot 10^{+00}$
FMINSURF		121	32	27	$1.000 \cdot 10^{+00}$	23	19	$1.000 \cdot 10^{+00}$	102	98	$1.000 \cdot 10^{+00}$	70	59	$1.000 \cdot 10^{+00}$
FREUROTH	*	100	9	10	$1.197 \cdot 10^{+04}$	9	10	$1.197 \cdot 10^{+04}$	14	15	$1.197 \cdot 10^{+04}$	14	15	$1.197 \cdot 10^{+04}$
GENHUMPS	*	10	10,402	9,802	$3.785 \cdot 10^{-12}$	11,624	10,931	$4.326 \cdot 10^{-13}$	5,083	4,434	$6.400 \cdot 10^{-13}$	7,075	6,449	$2.720 \cdot 10^{-14}$
GENROSE	*	100	107	88	$1.000 \cdot 10^{+00}$	90	83	$1.000 \cdot 10^{+00}$	130	116	$1.000 \cdot 10^{+00}$	123	113	$1.000 \cdot 10^{+00}$
GENROSEB		500	460	369	$1.000 \cdot 10^{+00}$	327	325	$1.000 \cdot 10^{+00}$	585	505	$1.000 \cdot 10^{+00}$	498	473	$1.000 \cdot 10^{+00}$
GROWTHLS	*	3	96	78	$1.004 \cdot 10^{+00}$	79	72	$1.004 \cdot 10^{+00}$	183	172	$1.004 \cdot 10^{+00}$	171	163	$1.004 \cdot 10^{+00}$

continued on next page...

Name	LS	n	Moré-Sorensen						Steihaug-Toint					
			BTR			RTR			BTR			RTR		
			iter	#g	f	iter	#g	f	iter	#g	f	iter	#g	f
GULF	*	3	30	28	$1.799 \cdot 10^{-17}$	32	30	$3.619 \cdot 10^{-14}$	40	38	$3.455 \cdot 10^{-13}$	44	43	$3.242 \cdot 10^{-09}$
HAIRY		2	64	57	$2.000 \cdot 10^{+01}$	116	107	$2.000 \cdot 10^{+01}$	96	84	$2.000 \cdot 10^{+01}$	91	86	$2.000 \cdot 10^{+01}$
HATFLDD	*	3	20	20	$6.615 \cdot 10^{-08}$	20	20	$6.615 \cdot 10^{-08}$	18	18	$6.694 \cdot 10^{-08}$	18	18	$6.694 \cdot 10^{-08}$
HATFLDE	*	3	21	21	$5.120 \cdot 10^{-07}$	20	20	$5.120 \cdot 10^{-07}$	17	17	$5.120 \cdot 10^{-07}$	17	17	$5.120 \cdot 10^{-07}$
HEART6LS	*	6	667	642	$4.411 \cdot 10^{-26}$	1,039	1,019	$2.119 \cdot 10^{-24}$	1,528	1,498	$7.291 \cdot 10^{-13}$	1,593	1,583	$1.597 \cdot 10^{-12}$
HEART8LS	*	8	112	95	$4.636 \cdot 10^{-17}$	102	88	$1.751 \cdot 10^{-13}$	152	143	$2.052 \cdot 10^{-20}$	159	154	$3.815 \cdot 10^{-14}$
HELIX	*	3	11	11	$5.659 \cdot 10^{-23}$	8	8	$4.960 \cdot 10^{-13}$	20	19	$7.740 \cdot 10^{-15}$	15	14	$1.848 \cdot 10^{-15}$
HIELOW		3	11	10	$8.742 \cdot 10^{+02}$	8	8	$8.742 \cdot 10^{+02}$	13	12	$8.742 \cdot 10^{+02}$	12	11	$8.742 \cdot 10^{+02}$
HILBERTA		2	3	4	$2.054 \cdot 10^{-33}$	3	4	$2.054 \cdot 10^{-33}$	3	4	$1.855 \cdot 10^{-30}$	3	4	$1.855 \cdot 10^{-30}$
HILBERTB		10	3	4	$1.884 \cdot 10^{-29}$	3	4	$1.884 \cdot 10^{-29}$	7	8	$2.223 \cdot 10^{-14}$	7	8	$2.223 \cdot 10^{-14}$
HIMMELBB		2	10	9	$5.174 \cdot 10^{-16}$	10	8	$1.242 \cdot 10^{-20}$	19	19	$1.755 \cdot 10^{-11}$	19	19	$1.755 \cdot 10^{-11}$
HIMMELBF	*	4	276	274	$3.186 \cdot 10^{+02}$	94	92	$3.186 \cdot 10^{+02}$	358	356	$3.186 \cdot 10^{+02}$	353	315	$3.186 \cdot 10^{+02}$
HIMMELBG		2	5	6	$9.033 \cdot 10^{-12}$	5	6	$9.033 \cdot 10^{-12}$	7	7	$1.731 \cdot 10^{-15}$	7	7	$1.731 \cdot 10^{-15}$
HIMMELBH		2	4	5	$-1.000 \cdot 10^{+00}$	4	5	$-1.000 \cdot 10^{+00}$	4	5	$-1.000 \cdot 10^{+00}$	4	5	$-1.000 \cdot 10^{+00}$
HUMPS	*	2	2,690	2,503	$1.098 \cdot 10^{-12}$	6,856	6,604	$2.403 \cdot 10^{-13}$	2,606	2,243	$6.092 \cdot 10^{-14}$	6,265	6,038	$6.537 \cdot 10^{-11}$
JENSMP		2	9	10	$1.244 \cdot 10^{+02}$	9	10	$1.244 \cdot 10^{+02}$	9	10	$1.244 \cdot 10^{+02}$	9	10	$1.244 \cdot 10^{+02}$
KOWOSB	*	4	11	10	$3.078 \cdot 10^{-04}$	11	10	$3.078 \cdot 10^{-04}$	12	12	$3.078 \cdot 10^{-04}$	12	11	$3.078 \cdot 10^{-04}$
LIARWHD	*	100	12	13	$5.568 \cdot 10^{-14}$	12	13	$5.568 \cdot 10^{-14}$	14	15	$2.468 \cdot 10^{-15}$	14	15	$2.468 \cdot 10^{-15}$
LOGHAIRY		2	2,734	2,676	$1.823 \cdot 10^{-01}$	9,091	8,167	$1.823 \cdot 10^{-01}$	4,871	4,132	$5.128 \cdot 10^{+00}$	7,612	6,953	$1.823 \cdot 10^{-01}$
MANCINO	*	100	14	15	$1.506 \cdot 10^{-21}$	16	16	$4.061 \cdot 10^{-19}$	20	21	$1.449 \cdot 10^{-21}$	20	21	$1.449 \cdot 10^{-21}$
MARATOSB		2	699	673	$-1.000 \cdot 10^{+00}$	680	667	$-1.000 \cdot 10^{+00}$	1,882	1,726	$-1.000 \cdot 10^{+00}$	1,547	1,493	$-1.000 \cdot 10^{+00}$
MEXHAT		2	32	30	$-4.001 \cdot 10^{-02}$	31	30	$-4.001 \cdot 10^{-02}$	19	20	$-4.001 \cdot 10^{-02}$	19	20	$-4.001 \cdot 10^{-02}$
MEYER3	*	3	481	441	$8.795 \cdot 10^{+01}$	416	381	$8.795 \cdot 10^{+01}$	686	680	$8.851 \cdot 10^{+01}$	693	688	$8.819 \cdot 10^{+01}$
MODBEALE		200	10	11	$7.824 \cdot 10^{-21}$	10	11	$7.824 \cdot 10^{-21}$	14	15	$3.111 \cdot 10^{-15}$	14	15	$3.111 \cdot 10^{-15}$
MOREBV	*	100	1	2	$7.887 \cdot 10^{-10}$	1	2	$7.887 \cdot 10^{-10}$	138	139	$2.140 \cdot 10^{-07}$	138	139	$2.140 \cdot 10^{-07}$
MSQRTALS	*	100	20	18	$2.677 \cdot 10^{-17}$	19	17	$7.470 \cdot 10^{-10}$	20	19	$4.032 \cdot 10^{-11}$	20	19	$4.032 \cdot 10^{-11}$
MSQRTBLS	*	100	16	14	$1.886 \cdot 10^{-17}$	16	14	$9.418 \cdot 10^{-14}$	21	20	$4.133 \cdot 10^{-14}$	21	20	$4.133 \cdot 10^{-14}$
NONCVXU2		100	53	47	$2.318 \cdot 10^{+02}$	49	41	$2.324 \cdot 10^{+02}$	45	40	$2.324 \cdot 10^{+02}$	41	34	$2.324 \cdot 10^{+02}$
NONCVXUN		100	42	38	$2.317 \cdot 10^{+02}$	41	36	$2.329 \cdot 10^{+02}$	44	40	$2.317 \cdot 10^{+02}$	41	34	$2.323 \cdot 10^{+02}$
NONDIA	*	100	6	7	$1.495 \cdot 10^{-18}$	6	7	$1.495 \cdot 10^{-18}$	10	11	$6.598 \cdot 10^{-15}$	10	11	$6.598 \cdot 10^{-15}$
NONDQUAR		100	15	16	$2.699 \cdot 10^{-09}$	15	16	$2.699 \cdot 10^{-09}$	110	84	$2.198 \cdot 10^{-06}$	97	86	$1.973 \cdot 10^{-06}$
OSBORNEA	*	5	37	32	$5.465 \cdot 10^{-05}$	30	27	$5.465 \cdot 10^{-05}$	64	59	$5.472 \cdot 10^{-05}$	82	79	$5.465 \cdot 10^{-05}$
OSBORNEB	*	11	21	19	$4.014 \cdot 10^{-02}$	21	19	$4.014 \cdot 10^{-02}$	22	22	$4.014 \cdot 10^{-02}$	22	22	$4.014 \cdot 10^{-02}$
OSCIPATH		8	2,035	1,734	$1.747 \cdot 10^{-05}$	2,015	1,804	$1.481 \cdot 10^{-05}$	3,020	2,625	$3.366 \cdot 10^{-05}$	2,670	2,488	$4.394 \cdot 10^{-05}$

continued on next page...

Name	LS	n	Moré-Sorensen						Steihaug-Toint					
			BTR			RTR			BTR			RTR		
			iter	#g	f	iter	#g	f	iter	#g	f	iter	#g	f
PALMER1C		8	7	8	$9.761 \cdot 10^{-02}$	7	8	$9.761 \cdot 10^{-02}$	–	–	$9.765 \cdot 10^{-02}$	–	–	$9.765 \cdot 10^{-02}$
PALMER1D		7	7	8	$6.527 \cdot 10^{-01}$	7	8	$6.527 \cdot 10^{-01}$	23	24	$6.527 \cdot 10^{-01}$	23	24	$6.527 \cdot 10^{-01}$
PALMER2C		8	6	7	$1.437 \cdot 10^{-02}$	6	7	$1.437 \cdot 10^{-02}$	3,161	3,162	$1.437 \cdot 10^{-02}$	3,161	3,162	$1.437 \cdot 10^{-02}$
PALMER3C		8	6	7	$1.954 \cdot 10^{-02}$	6	7	$1.954 \cdot 10^{-02}$	1,784	1,785	$1.954 \cdot 10^{-02}$	1,784	1,785	$1.954 \cdot 10^{-02}$
PALMER4C		8	7	8	$5.031 \cdot 10^{-02}$	7	8	$5.031 \cdot 10^{-02}$	1,538	1,539	$5.031 \cdot 10^{-02}$	1,538	1,539	$5.031 \cdot 10^{-02}$
PALMER5C	*	6	5	6	$2.128 \cdot 10^{+00}$	5	6	$2.128 \cdot 10^{+00}$	9	10	$2.128 \cdot 10^{+00}$	9	10	$2.128 \cdot 10^{+00}$
PALMER6C	*	8	7	8	$1.639 \cdot 10^{-02}$	7	8	$1.639 \cdot 10^{-02}$	165	166	$1.639 \cdot 10^{-02}$	165	166	$1.639 \cdot 10^{-02}$
PALMER7C	*	8	9	10	$6.020 \cdot 10^{-01}$	9	10	$6.020 \cdot 10^{-01}$	6,810	5,734	$6.020 \cdot 10^{-01}$	4,456	3,946	$6.020 \cdot 10^{-01}$
PALMER8C	*	8	8	9	$1.598 \cdot 10^{-01}$	8	9	$1.598 \cdot 10^{-01}$	197	198	$1.598 \cdot 10^{-01}$	197	198	$1.598 \cdot 10^{-01}$
PENALTY1	*	100	45	44	$9.025 \cdot 10^{-04}$	45	44	$9.025 \cdot 10^{-04}$	44	41	$9.026 \cdot 10^{-04}$	48	44	$9.025 \cdot 10^{-04}$
PENALTY2	*	100	19	20	$9.710 \cdot 10^{+04}$	19	20	$9.710 \cdot 10^{+04}$	19	20	$9.710 \cdot 10^{+04}$	19	20	$9.710 \cdot 10^{+04}$
PFIT1LS	*	3	325	287	$1.573 \cdot 10^{-16}$	294	280	$3.086 \cdot 10^{-15}$	365	350	$4.851 \cdot 10^{-07}$	384	379	$4.351 \cdot 10^{-07}$
PFIT2LS	*	3	114	98	$3.622 \cdot 10^{-15}$	90	84	$3.423 \cdot 10^{-20}$	133	128	$1.962 \cdot 10^{-08}$	161	158	$7.535 \cdot 10^{-09}$
PFIT3LS	*	3	144	125	$4.464 \cdot 10^{-19}$	126	116	$3.643 \cdot 10^{-14}$	222	211	$1.252 \cdot 10^{-08}$	226	221	$2.479 \cdot 10^{-09}$
PFIT4LS	*	3	241	218	$3.414 \cdot 10^{-20}$	232	223	$8.814 \cdot 10^{-23}$	401	390	$6.139 \cdot 10^{-10}$	495	491	$7.142 \cdot 10^{-10}$
POWELLSG		4	15	16	$4.633 \cdot 10^{-09}$	15	16	$4.633 \cdot 10^{-09}$	15	16	$1.273 \cdot 10^{-08}$	15	16	$1.273 \cdot 10^{-08}$
POWER		100	24	25	$1.182 \cdot 10^{-09}$	24	25	$1.182 \cdot 10^{-09}$	25	26	$1.669 \cdot 10^{-09}$	25	26	$1.669 \cdot 10^{-09}$
QUARTC		100	29	30	$2.806 \cdot 10^{-08}$	29	30	$2.806 \cdot 10^{-08}$	29	30	$3.590 \cdot 10^{-08}$	29	30	$3.590 \cdot 10^{-08}$
ROSENBR	*	2	30	26	$7.149 \cdot 10^{-15}$	28	26	$6.021 \cdot 10^{-13}$	34	30	$2.823 \cdot 10^{-14}$	34	31	$5.798 \cdot 10^{-11}$
S308	*	2	13	12	$7.732 \cdot 10^{-01}$	13	12	$7.732 \cdot 10^{-01}$	9	10	$7.732 \cdot 10^{-01}$	9	10	$7.732 \cdot 10^{-01}$
SBRYBND	*	100	46	37	$2.562 \cdot 10^{-22}$	46	37	$9.126 \cdot 10^{-15}$	–	–	$2.653 \cdot 10^{+01}$	–	–	$2.546 \cdot 10^{+01}$
SCHMVETT		100	4	5	$-2.940 \cdot 10^{+02}$	4	5	$-2.940 \cdot 10^{+02}$	6	7	$-2.940 \cdot 10^{+02}$	6	7	$-2.940 \cdot 10^{+02}$
SCOSINE		100	–	–	$-9.884 \cdot 10^{+01}$	97	90	$-9.900 \cdot 10^{+01}$	–	–	$-9.731 \cdot 10^{+01}$	–	–	$-9.338 \cdot 10^{+01}$
SCURLY10	*	100	39	35	$-1.003 \cdot 10^{+04}$	46	42	$-1.003 \cdot 10^{+04}$	–	–	$-1.001 \cdot 10^{+04}$	–	–	$-1.001 \cdot 10^{+04}$
SCURLY20	*	100	34	30	$-1.003 \cdot 10^{+04}$	37	33	$-1.003 \cdot 10^{+04}$	–	–	$-1.003 \cdot 10^{+04}$	–	–	$-1.003 \cdot 10^{+04}$
SCURLY30	*	100	35	31	$-1.003 \cdot 10^{+04}$	35	31	$-1.003 \cdot 10^{+04}$	–	–	$-1.002 \cdot 10^{+04}$	–	–	$-1.002 \cdot 10^{+04}$
SENSORS	*	100	21	21	$-1.967 \cdot 10^{+03}$	24	23	$-1.967 \cdot 10^{+03}$	20	20	$-2.025 \cdot 10^{+03}$	24	22	$-2.025 \cdot 10^{+03}$
SINEVAL	*	2	53	46	$1.974 \cdot 10^{-25}$	58	52	$3.381 \cdot 10^{-36}$	107	93	$3.619 \cdot 10^{-18}$	80	73	$1.445 \cdot 10^{-21}$
SINQUAD		100	9	10	$-4.006 \cdot 10^{+03}$	9	10	$-4.006 \cdot 10^{+03}$	14	14	$-4.006 \cdot 10^{+03}$	11	12	$-4.006 \cdot 10^{+03}$
SISSER		2	12	13	$1.066 \cdot 10^{-08}$	12	13	$1.066 \cdot 10^{-08}$	12	13	$1.214 \cdot 10^{-08}$	12	13	$1.214 \cdot 10^{-08}$
SNAIL		2	61	61	$9.370 \cdot 10^{-13}$	59	60	$1.212 \cdot 10^{-14}$	72	72	$8.616 \cdot 10^{-17}$	62	63	$3.640 \cdot 10^{-18}$
SPARSINE		100	37	27	$9.379 \cdot 10^{-16}$	30	22	$2.873 \cdot 10^{-16}$	10	11	$1.716 \cdot 10^{-15}$	10	11	$1.716 \cdot 10^{-15}$
SPARSQUR		100	16	17	$1.480 \cdot 10^{-08}$	16	17	$1.480 \cdot 10^{-08}$	16	17	$1.987 \cdot 10^{-08}$	16	17	$1.987 \cdot 10^{-08}$
SPMSRTLS	*	100	14	13	$1.259 \cdot 10^{-13}$	12	11	$6.136 \cdot 10^{-12}$	13	13	$4.666 \cdot 10^{-14}$	13	13	$4.666 \cdot 10^{-14}$

continued on next page...

Name	LS	n	Moré-Sorensen						Steihaug-Toint					
			BTR			RTR			BTR			RTR		
			iter	#g	f	iter	#g	f	iter	#g	f	iter	#g	f
SROSENBR	*	100	6	7	$8.899 \cdot 10^{-28}$	6	7	$8.899 \cdot 10^{-28}$	8	9	$2.608 \cdot 10^{-19}$	8	9	$2.608 \cdot 10^{-19}$
TOINTGOR		50	9	10	$1.374 \cdot 10^{+03}$	9	10	$1.374 \cdot 10^{+03}$	11	12	$1.374 \cdot 10^{+03}$	11	12	$1.374 \cdot 10^{+03}$
TOINTGSS		100	17	15	$1.010 \cdot 10^{+01}$	13	13	$1.020 \cdot 10^{+01}$	12	12	$1.010 \cdot 10^{+01}$	12	12	$1.010 \cdot 10^{+01}$
TOINTPSP		50	22	20	$2.256 \cdot 10^{+02}$	30	28	$2.256 \cdot 10^{+02}$	47	38	$2.256 \cdot 10^{+02}$	58	50	$2.256 \cdot 10^{+02}$
TQUARTIC	*	100	14	13	$2.677 \cdot 10^{-24}$	15	13	$1.497 \cdot 10^{-17}$	15	15	$5.309 \cdot 10^{-15}$	15	15	$5.309 \cdot 10^{-15}$
VARDIM		200	29	30	$2.908 \cdot 10^{-24}$	29	30	$2.908 \cdot 10^{-24}$	29	30	$2.068 \cdot 10^{-25}$	29	30	$2.068 \cdot 10^{-25}$
VAREIGVL	*	50	15	13	$4.712 \cdot 10^{-09}$	16	14	$1.355 \cdot 10^{-10}$	13	14	$2.271 \cdot 10^{-10}$	13	14	$2.271 \cdot 10^{-10}$
VIBRBEAM	*	8	49	39	$1.749 \cdot 10^{+00}$	51	40	$1.749 \cdot 10^{+00}$	668	669	$1.565 \cdot 10^{-01}$	960	956	$1.565 \cdot 10^{-01}$
WATSON	*	12	14	14	$8.154 \cdot 10^{-07}$	13	13	$3.907 \cdot 10^{-08}$	12	13	$1.597 \cdot 10^{-07}$	12	13	$1.597 \cdot 10^{-07}$
WOODS	*	4	52	44	$4.641 \cdot 10^{-15}$	53	47	$5.156 \cdot 10^{-17}$	69	59	$2.067 \cdot 10^{-13}$	60	54	$3.828 \cdot 10^{-17}$
YFITU	*	3	54	48	$6.686 \cdot 10^{-13}$	50	46	$6.670 \cdot 10^{-13}$	85	77	$2.296 \cdot 10^{-08}$	79	75	$1.017 \cdot 10^{-08}$

Table A.1 — *Results of the retrospective trust-region method.*

A.2 Sparse Hessian approximation

In Table A.2, we report the CPU time, the equivalent number of function (#f), gradient (#g) and Hessian (#B) evaluations. The symbol "–" indicates that the iteration limit (fixed at 10,000) or the time limit (fixed at 1 hour) was exceeded.

A.3 Multilevel L-BFGS method

In Tables A.3 to A.16, we display the details of our numerical results (each problem having its own table of results). The first column indicates the chosen conjugacy factor β (see Section 2.6.3): either the quasi-Newton (QN) choice, or the Dai-Yuan-Hestenes-Stiefel choice (DYHS), or the Hager-Zhang choice (HZ). The second column indicates the linesearch algorithm that was used (see Section 2.5): either the Hager-Zhang one (HZ), or the Dennis-Schnabel one (DS). The third and fourth columns indicate the strategies used to select and order, respectively, the secant pairs (see Section 6.1), while the fifth column gives the value of parameter τ in equation (6.10). The sixth column indicates the execution return status:

- 0: the algorithm ran successfully;
- 4: the Hager-Zhang linesearch failed because of too many secant steps;
- 8: the Hager-Zhang linesearch failed;
- 12: the Dennis-Schnabel linesearch failed;
- -7: the function value became `-Inf`.

The last three columns indicates the number of iterations (iter), function evaluations (#f), and gradient evaluations (#g), respectively.

Problem	n	LTS-O				LTS				S-PSB			
		CPU	#f	#g	#B	CPU	#f	#g	#B	CPU	#f	#g	#B
P2D	261,121	0.88	1.88	2.47	0.08	0.80	1.88	1.96	0.08	2.43	5.24	3.55	1.07
P2D	1,046,529	3.69	1.70	2.35	0.08	3.32	1.70	1.84	0.08	5.83	2.48	2.09	0.34
P2D	4,190,209	13.33	1.43	1.59	0.02	11.54	1.43	1.46	0.02	17.70	1.70	1.59	0.12
P3D	29,791	1.86	6.19	21.96	1.14	1.10	6.19	9.45	1.14	12.36	88.61	53.73	25.74
P3D	250,047	18.31	6.03	21.87	1.14	10.52	6.03	9.31	1.14	143.50	119.03	59.15	28.44
DEPT	261,121	0.58	1.67	2.15	0.08	0.55	1.67	1.90	0.08	1.18	3.91	2.49	0.54
DEPT	1,046,529	2.07	1.47	1.57	0.02	2.04	1.47	1.51	0.02	3.55	2.20	1.81	0.23
DEPT	4,190,209	8.21	1.38	1.52	0.02	7.80	1.38	1.46	0.02	13.39	2.04	1.60	0.12
DPJB	261,121	0.69	1.87	2.48	0.08	0.62	1.87	2.05	0.08	2.47	6.77	4.65	1.62
DPJB	1,046,529	2.73	1.63	2.31	0.08	2.57	1.63	1.89	0.08	7.56	4.91	3.08	0.83
DPJB	4,190,209	11.28	1.46	1.61	0.02	10.11	1.46	1.50	0.02	31.38	2.86	2.25	0.45
DODC	261,121	2.41	4.54	13.68	1.13	14.39	42.78	90.04	15.01	4.68	13.63	6.91	2.71
DODC	1,046,529	6.33	2.40	5.75	0.40	65.05	34.87	88.18	14.57	16.18	7.54	5.65	2.21
DODC	4,190,209	24.72	2.23	4.45	0.29	186.52	18.71	40.99	6.65	38.18	4.16	2.81	0.81
MINS-SB	65,025	10.70	86.05	295.36	24.28	9.43	99.05	202.89	28.28	45.65	914.76	518.48	45.93
MINS-SB	261,121	53.04	99.15	364.73	28.70	41.01	104.52	205.17	27.70	234.72	987.57	617.88	68.79
MINS-SB	1,046,529	227.83	99.26	375.90	29.42	173.90	102.67	209.02	30.86	1,429.00	1,589.16	912.60	99.64
MINS-OB	65,025	3.96	29.22	111.59	10.10	2.93	29.22	61.04	10.10	13.33	141.54	82.13	39.73
MINS-OB	261,121	51.16	90.12	398.27	33.65	28.13	60.37	156.01	26.15	57.59	154.79	86.25	41.79
MINS-OB	1,046,529	184.29	75.51	352.23	29.47	114.60	71.76	158.60	26.47	174.74	113.10	63.72	30.53
MINS-DMSA	65,025	3.98	27.09	98.37	8.91	21.19	195.10	522.04	87.02	11.79	103.27	70.95	34.15
MINS-DMSA	261,121	30.89	50.43	204.22	18.56	95.51	207.78	562.36	93.73	40.61	105.21	60.13	28.73
MINS-DMSA	1,046,529	119.48	49.11	191.30	17.39	652.20	334.14	934.21	155.70	128.06	83.01	48.96	23.52
IGNISC	261,121	5.57	17.10	49.60	4.47	0.82	1.97	4.22	0.34	7.02	17.25	11.63	5.41
IGNISC	1,046,529	35.55	22.88	80.71	7.67	3.63	1.99	4.22	0.33	29.84	21.04	11.69	4.53
DSSC	261,121	0.80	1.68	2.33	0.08	0.71	1.68	1.90	0.08	1.77	3.79	2.84	0.71
DSSC	1,046,529	2.87	1.48	1.62	0.02	2.83	1.48	1.51	0.02	4.55	2.08	1.84	0.24
DSSC	4,190,209	11.78	1.39	1.56	0.02	10.85	1.39	1.46	0.02	14.65	1.66	1.51	0.08
BRATU	65,025	0.35	2.57	5.26	0.33	0.25	2.57	3.20	0.33	1.01	9.99	6.44	2.39
BRATU	261,121	0.85	1.73	2.36	0.08	0.78	1.73	1.84	0.08	2.05	4.25	3.31	0.95
BRATU	1,046,529	3.58	1.55	2.25	0.08	3.25	1.55	1.75	0.08	5.79	2.67	2.12	0.35
MEMBR	261,121	96.29	146.65	554.56	46.75	75.95	191.72	256.87	51.13	163.53	686.03	377.24	44.12
MEMBR	1,046,529	163.75	45.57	166.86	14.16	152.82	54.98	71.55	14.06	661.63	644.41	330.95	12.74
MEMBR	4,190,209	380.26	16.36	55.48	4.70	452.74	16.15	20.61	3.86	1,405.36	262.67	134.00	3.23
NCCS	7,938	0.29	11.52	60.82	1.34	0.13	11.52	26.88	1.34	0.14	6.28	4.12	3.00
NCCS	32,258	0.80	8.47	61.00	1.34	0.40	8.47	24.52	1.34	0.19	2.57	2.03	0.75
NCCS	130,050	11.37	12.02	264.70	6.33	1.53	7.02	23.58	1.33	–	–	–	–
NCCO	32,258	0.05	1.34	1.37	0.00	0.04	1.34	1.37	0.00	0.16	2.57	2.03	0.75
NCCO	130,050	11.51	9.33	289.34	7.00	0.83	3.33	16.34	1.00	2.57	22.64	2.51	0.19
NCCO	522,242	12.03	3.33	73.34	1.75	1.38	1.83	5.09	0.25	–	–	–	–
MOREBV	65,025	9.74	94.71	55.30	0.33	8.76	94.71	51.07	0.33	20.06	264.23	197.74	11.71
MOREBV	261,121	29.62	43.44	24.25	0.08	28.87	43.44	23.20	0.08	46.17	145.85	101.13	6.03
MOREBV	1,046,529	137.13	19.41	12.20	0.08	121.24	19.41	11.06	0.08	450.78	459.46	236.22	2.76

continued on next page...

Problem	n	PS-PSB				UP-BFGS				BP-BFGS			
		CPU	#f	#g	#B	CPU	#f	#g	#B	CPU	#f	#g	#B
P2D	261,121	2.71	5.32	3.76	1.17	12.90	22.96	15.50	7.04	127.24	22.87	15.44	7.01
P2D	1,046,529	6.28	2.60	2.19	0.39	34.17	12.46	8.67	3.63	243.75	12.29	8.52	3.55
P2D	4,190,209	19.19	1.71	1.59	0.12	–	–	–	–	–	–	–	–
P3D	29,791	12.67	92.79	51.12	24.43	11.30	74.84	48.45	23.11	152.70	78.63	51.04	24.47
P3D	250,047	184.30	139.67	84.77	41.24	130.35	90.27	57.34	27.53	–	–	–	–
DEPT	261,121	1.38	4.28	2.81	0.70	10.41	27.58	16.91	7.78	74.94	27.85	17.24	7.91
DEPT	1,046,529	3.73	2.19	1.81	0.23	27.96	11.06	7.07	2.86	134.64	12.65	7.79	3.22
DEPT	4,190,209	15.85	2.55	1.86	0.19	93.50	5.32	3.58	1.11	–	–	–	–
DPJB	261,121	2.38	6.17	4.13	1.36	6.56	13.99	9.22	3.90	84.27	15.74	10.31	4.45
DPJB	1,046,529	8.45	5.31	3.35	0.97	25.65	11.30	7.63	3.11	234.51	11.37	7.77	3.18
DPJB	4,190,209	22.73	2.52	1.95	0.30	–	–	–	–	–	–	–	–
DDDC	261,121	4.68	14.11	6.46	2.40	68.43	165.18	94.23	46.30	668.77	255.94	137.65	68.17
DDDC	1,046,529	13.98	6.35	4.56	1.69	537.42	315.22	166.44	82.52	–	–	–	–
DDDC	4,190,209	42.76	4.46	2.77	0.80	–	–	–	–	–	–	–	–
MINS-SB	65,025	31.61	555.84	362.48	39.45	84.65	653.59	582.71	199.22	947.50	840.55	781.92	338.50
MINS-SB	261,121	201.77	894.33	539.26	55.88	2,191.21	3,359.43	3,306.73	1,579.73	–	–	–	–
MINS-SB	1,046,529	1,279.08	1,382.69	804.51	96.06	3,108.23	1,117.39	1,074.15	525.82	–	–	–	–
MINS-OB	65,025	15.85	173.29	98.77	48.06	22.37	178.39	117.47	57.41	347.34	187.87	122.08	59.71
MINS-OB	261,121	58.44	151.90	87.12	42.23	79.25	149.86	100.29	48.81	1,454.49	164.08	108.05	52.69
MINS-OB	1,046,529	213.70	160.04	83.40	40.37	482.94	276.23	162.65	79.99	–	–	–	–
MINS-DMSA	65,025	10.60	97.58	61.29	29.31	34.71	277.40	179.19	88.27	463.76	279.24	180.39	88.86
MINS-DMSA	261,121	33.50	87.02	51.99	24.66	104.21	207.93	134.25	65.82	1,460.76	227.11	146.39	71.86
MINS-DMSA	1,046,529	126.76	92.12	54.96	26.15	329.05	157.77	99.53	48.62	–	–	–	–
IGNISC	261,121	6.02	14.94	9.36	3.90	6.33	14.17	9.12	3.79	123.64	14.30	8.86	3.63
IGNISC	1,046,529	24.18	14.48	8.19	3.59	35.11	17.43	10.26	4.40	588.85	17.33	10.35	4.38
DSSC	261,121	1.81	3.49	2.74	0.66	10.51	19.99	12.32	5.45	77.21	20.38	12.42	5.50
DSSC	1,046,529	4.88	2.08	1.86	0.25	24.42	8.48	5.42	2.04	126.81	8.38	5.32	1.98
DSSC	4,190,209	15.95	1.72	1.55	0.10	–	–	–	–	–	–	–	–
BRATU	65,025	0.70	7.10	4.36	1.35	3.78	32.25	20.28	9.31	36.87	31.46	19.82	9.08
BRATU	261,121	1.92	3.84	2.91	0.75	10.31	21.68	12.50	5.54	73.41	15.99	9.85	4.22
BRATU	1,046,529	5.20	2.48	1.91	0.25	35.82	13.49	7.77	3.18	189.15	11.00	6.62	2.60
MEMBR	261,121	96.66	234.36	128.07	50.20	171.86	391.21	197.86	98.27	645.75	387.99	198.45	98.55
MEMBR	1,046,529	242.78	159.23	79.63	13.75	351.39	107.60	54.81	26.71	1,045.92	110.47	56.86	27.73
MEMBR	4,190,209	651.50	80.32	39.02	3.61	–	–	–	–	–	–	–	–
NCCS	7,938	0.24	10.33	5.94	5.54	0.04	2.15	1.88	0.61	51.26	309.39	166.18	85.42
NCCS	32,258	0.35	3.58	2.49	1.39	0.08	1.58	1.47	0.15	411.80	415.97	226.73	138.35
NCCS	130,050	–	–	–	–	–	–	–	–	–	–	–	–
NCCD	32,258	0.40	3.58	2.49	1.39	0.08	1.58	1.47	0.15	0.09	1.58	1.47	0.15
NCCD	130,050	2.69	22.90	2.62	0.35	3.88	22.40	2.37	0.04	3.88	22.40	2.37	0.04
NCCD	522,242	–	–	–	–	–	–	–	–	–	–	–	–
MOREBV	65,025	7.91	36.80	28.55	12.72	84.49	465.10	253.53	128.27	620.03	361.73	211.12	105.11
MOREBV	261,121	26.19	26.23	22.09	8.51	271.05	362.46	187.63	93.71	1,802.63	342.01	180.34	89.62
MOREBV	1,046,529	68.21	14.34	11.33	3.95	463.23	100.26	52.25	25.57	2,468.63	94.53	50.04	24.37

Table A.2 — *Comparison of Hessian approximation methods in the context of optimization problems arising from discretization.*

β	linesearch	select	order	τ	status	iter	#f	#g
QN	HZ	LBFGS	—	1.000	4	512	1,107	1,619
QN	HZ	Full	Coarse	1.000	0	259	355	614
QN	HZ	Full	Coarse	0.999	0	179	257	436
QN	HZ	Full	Fine	1.000	0	206	282	488
QN	HZ	Full	Fine	0.999	0	168	225	393
QN	HZ	Local	Coarse	1.000	0	257	368	625
QN	HZ	Local	Coarse	0.999	0	273	403	676
QN	HZ	Local	Fine	1.000	0	225	332	557
QN	HZ	Local	Fine	0.999	0	224	351	575
QN	HZ	Mless	Coarse	1.000	0	388	551	939
QN	HZ	Mless	Coarse	0.999	0	344	474	818
QN	HZ	Mless	Fine	1.000	0	167	220	387
QN	HZ	Mless	Fine	0.999	0	225	307	532
QN	DS	LBFGS	—	1.000	0	635	647	636
QN	DS	Full	Coarse	1.000	0	239	297	240
QN	DS	Full	Coarse	0.999	0	215	261	216
QN	DS	Full	Fine	1.000	0	188	232	189
QN	DS	Full	Fine	0.999	0	148	177	149
QN	DS	Local	Coarse	1.000	0	258	313	259
QN	DS	Local	Coarse	0.999	0	188	225	189
QN	DS	Local	Fine	1.000	0	218	272	219
QN	DS	Local	Fine	0.999	0	222	265	223
QN	DS	Mless	Coarse	1.000	0	364	460	366
QN	DS	Mless	Coarse	0.999	0	357	440	359
QN	DS	Mless	Fine	1.000	0	211	276	212
QN	DS	Mless	Fine	0.999	0	194	248	195
DYHS	HZ	LBFGS	—	1.000	0	184	369	185
DYHS	HZ	Full	Coarse	1.000	0	158	317	160
DYHS	HZ	Full	Coarse	0.999	0	162	325	165
DYHS	HZ	Full	Fine	1.000	0	162	325	164
DYHS	HZ	Full	Fine	0.999	0	125	251	126
DYHS	HZ	Local	Coarse	1.000	0	243	487	249
DYHS	HZ	Local	Coarse	0.999	0	192	385	198
DYHS	HZ	Local	Fine	1.000	0	203	407	210
DYHS	HZ	Local	Fine	0.999	0	171	343	179
DYHS	HZ	Mless	Coarse	1.000	0	487	975	500
DYHS	HZ	Mless	Coarse	0.999	0	200	401	214
DYHS	HZ	Mless	Fine	1.000	0	183	367	203
DYHS	HZ	Mless	Fine	0.999	0	150	301	169
DYHS	DS	LBFGS	—	1.000	0	662	1,328	663
DYHS	DS	Full	Coarse	1.000	0	236	527	237
DYHS	DS	Full	Coarse	0.999	0	198	445	199
DYHS	DS	Full	Fine	1.000	0	199	446	200
DYHS	DS	Full	Fine	0.999	0	135	298	136
DYHS	DS	Local	Coarse	1.000	0	231	507	232
DYHS	DS	Local	Coarse	0.999	0	165	372	166
DYHS	DS	Local	Fine	1.000	0	215	470	216
DYHS	DS	Local	Fine	0.999	0	159	352	160
DYHS	DS	Mless	Coarse	1.000	0	420	952	423
DYHS	DS	Mless	Coarse	0.999	0	358	798	363
DYHS	DS	Mless	Fine	1.000	0	172	398	173
DYHS	DS	Mless	Fine	0.999	0	200	455	203
HZ	HZ	LBFGS	—	1.000	0	184	369	185
HZ	HZ	Full	Coarse	1.000	0	194	389	198
HZ	HZ	Full	Coarse	0.999	0	145	291	146
HZ	HZ	Full	Fine	1.000	0	180	361	184
HZ	HZ	Full	Fine	0.999	0	125	251	126
HZ	HZ	Local	Coarse	1.000	0	181	363	185
HZ	HZ	Local	Coarse	0.999	0	216	433	221
HZ	HZ	Local	Fine	1.000	0	206	413	214
HZ	HZ	Local	Fine	0.999	0	163	327	167
HZ	HZ	Mless	Coarse	1.000	0	371	743	381
HZ	HZ	Mless	Coarse	0.999	0	271	543	284
HZ	HZ	Mless	Fine	1.000	0	152	305	174
HZ	HZ	Mless	Fine	0.999	0	177	355	201
HZ	DS	LBFGS	—	1.000	0	1,329	2,694	1,335
HZ	DS	Full	Coarse	1.000	0	237	534	238
HZ	DS	Full	Coarse	0.999	0	194	432	195
HZ	DS	Full	Fine	1.000	0	191	426	192
HZ	DS	Full	Fine	0.999	0	489	1,010	490
HZ	DS	Local	Coarse	1.000	0	229	518	230
HZ	DS	Local	Coarse	0.999	0	221	492	222
HZ	DS	Local	Fine	1.000	0	224	493	225
HZ	DS	Local	Fine	0.999	0	190	413	191
HZ	DS	Mless	Coarse	1.000	0	248	546	249
HZ	DS	Mless	Coarse	0.999	0	290	653	291
HZ	DS	Mless	Fine	1.000	0	253	571	255
HZ	DS	Mless	Fine	0.999	0	369	814	372

Table A.3 — *Results for problem* `DNT` *(level 8, 261,121 variables).*

β	linesearch	select	order	τ	status	iter	#f	#g
QN	HZ	LBFGS	—	1.000	0	992	1,983	2,975
QN	HZ	Full	Coarse	1.000	0	275	427	702
QN	HZ	Full	Coarse	0.999	0	249	381	630
QN	HZ	Full	Fine	1.000	0	249	407	656
QN	HZ	Full	Fine	0.999	0	228	344	572
QN	HZ	Local	Coarse	1.000	0	318	496	814
QN	HZ	Local	Coarse	0.999	0	252	401	653
QN	HZ	Local	Fine	1.000	0	240	383	623
QN	HZ	Local	Fine	0.999	0	257	421	678
QN	HZ	Mless	Coarse	1.000	0	282	423	705
QN	HZ	Mless	Coarse	0.999	0	189	276	465
QN	HZ	Mless	Fine	1.000	0	154	232	386
QN	HZ	Mless	Fine	0.999	0	171	258	429
QN	DS	LBFGS	—	1.000	0	729	748	730
QN	DS	Full	Coarse	1.000	0	234	287	235
QN	DS	Full	Coarse	0.999	0	197	233	198
QN	DS	Full	Fine	1.000	0	221	263	222
QN	DS	Full	Fine	0.999	0	254	306	255
QN	DS	Local	Coarse	1.000	0	175	222	176
QN	DS	Local	Coarse	0.999	0	221	269	222
QN	DS	Local	Fine	1.000	0	235	283	236
QN	DS	Local	Fine	0.999	0	258	316	259
QN	DS	Mless	Coarse	1.000	0	202	264	204
QN	DS	Mless	Coarse	0.999	0	195	257	198
QN	DS	Mless	Fine	1.000	0	179	231	180
QN	DS	Mless	Fine	0.999	0	211	281	216
DYHS	HZ	LBFGS	—	1.000	0	676	1,353	677
DYHS	HZ	Full	Coarse	1.000	0	330	661	341
DYHS	HZ	Full	Coarse	0.999	0	207	415	211
DYHS	HZ	Full	Fine	1.000	0	187	375	190
DYHS	HZ	Full	Fine	0.999	0	190	381	193
DYHS	HZ	Local	Coarse	1.000	0	294	589	301
DYHS	HZ	Local	Coarse	0.999	0	246	493	254
DYHS	HZ	Local	Fine	1.000	0	207	415	213
DYHS	HZ	Local	Fine	0.999	0	245	491	252
DYHS	HZ	Mless	Coarse	1.000	0	226	453	236
DYHS	HZ	Mless	Coarse	0.999	0	250	501	267
DYHS	HZ	Mless	Fine	1.000	0	137	275	151
DYHS	HZ	Mless	Fine	0.999	0	213	427	246
DYHS	DS	LBFGS	—	1.000	0	944	1,890	945
DYHS	DS	Full	Coarse	1.000	0	280	611	281
DYHS	DS	Full	Coarse	0.999	0	318	705	320
DYHS	DS	Full	Fine	1.000	0	245	537	246
DYHS	DS	Full	Fine	0.999	0	242	533	243
DYHS	DS	Local	Coarse	1.000	0	294	657	295
DYHS	DS	Local	Coarse	0.999	0	252	554	253
DYHS	DS	Local	Fine	1.000	0	208	455	209
DYHS	DS	Local	Fine	0.999	0	190	413	191
DYHS	DS	Mless	Coarse	1.000	0	283	648	284
DYHS	DS	Mless	Coarse	0.999	0	255	589	257
DYHS	DS	Mless	Fine	1.000	0	202	460	204
DYHS	DS	Mless	Fine	0.999	0	180	412	181
HZ	HZ	LBFGS	—	1.000	0	676	1,353	677
HZ	HZ	Full	Coarse	1.000	0	247	495	253
HZ	HZ	Full	Coarse	0.999	0	222	445	232
HZ	HZ	Full	Fine	1.000	0	201	403	203
HZ	HZ	Full	Fine	0.999	0	208	417	211
HZ	HZ	Local	Coarse	1.000	0	209	419	212
HZ	HZ	Local	Coarse	0.999	0	245	491	255
HZ	HZ	Local	Fine	1.000	0	227	455	234
HZ	HZ	Local	Fine	0.999	0	213	427	218
HZ	HZ	Mless	Coarse	1.000	0	211	423	224
HZ	HZ	Mless	Coarse	0.999	0	210	421	229
HZ	HZ	Mless	Fine	1.000	0	163	327	188
HZ	HZ	Mless	Fine	0.999	0	171	343	203
HZ	DS	LBFGS	—	1.000	0	875	1,796	877
HZ	DS	Full	Coarse	1.000	0	239	528	240
HZ	DS	Full	Coarse	0.999	0	263	584	264
HZ	DS	Full	Fine	1.000	0	231	523	232
HZ	DS	Full	Fine	0.999	0	228	496	229
HZ	DS	Local	Coarse	1.000	0	277	632	278
HZ	DS	Local	Coarse	0.999	0	551	1,142	553
HZ	DS	Local	Fine	1.000	0	222	494	223
HZ	DS	Local	Fine	0.999	0	524	1,103	529
HZ	DS	Mless	Coarse	1.000	0	247	554	248
HZ	DS	Mless	Coarse	0.999	0	235	532	237
HZ	DS	Mless	Fine	1.000	0	321	703	322
HZ	DS	Mless	Fine	0.999	0	851	1,754	857

Table A.4 — *Results for problem* P2D *(level 8, 261,121 variables).*

β	linesearch	select	order	τ	status	iter	#f	#g
QN	HZ	LBFGS	—	1.000	0	250	501	751
QN	HZ	Full	Coarse	1.000	0	125	185	310
QN	HZ	Full	Coarse	0.999	0	119	174	293
QN	HZ	Full	Fine	1.000	0	114	155	269
QN	HZ	Full	Fine	0.999	0	99	144	243
QN	HZ	Local	Coarse	1.000	0	97	151	248
QN	HZ	Local	Coarse	0.999	0	93	146	239
QN	HZ	Local	Fine	1.000	0	92	150	242
QN	HZ	Local	Fine	0.999	0	100	163	263
QN	HZ	Mless	Coarse	1.000	0	135	191	326
QN	HZ	Mless	Coarse	0.999	0	101	147	248
QN	HZ	Mless	Fine	1.000	0	82	120	202
QN	HZ	Mless	Fine	0.999	0	104	155	259
QN	DS	LBFGS	—	1.000	0	207	212	209
QN	DS	Full	Coarse	1.000	0	114	149	116
QN	DS	Full	Coarse	0.999	0	100	126	102
QN	DS	Full	Fine	1.000	0	91	116	93
QN	DS	Full	Fine	0.999	0	102	135	104
QN	DS	Local	Coarse	1.000	0	96	113	98
QN	DS	Local	Coarse	0.999	0	87	109	89
QN	DS	Local	Fine	1.000	0	98	111	100
QN	DS	Local	Fine	0.999	0	94	105	96
QN	DS	Mless	Coarse	1.000	0	99	127	101
QN	DS	Mless	Coarse	0.999	0	93	126	95
QN	DS	Mless	Fine	1.000	0	107	135	109
QN	DS	Mless	Fine	0.999	0	96	120	98
DYHS	HZ	LBFGS	—	1.000	0	153	307	154
DYHS	HZ	Full	Coarse	1.000	0	107	215	109
DYHS	HZ	Full	Coarse	0.999	0	109	219	113
DYHS	HZ	Full	Fine	1.000	0	92	185	95
DYHS	HZ	Full	Fine	0.999	0	87	175	90
DYHS	HZ	Local	Coarse	1.000	0	95	191	98
DYHS	HZ	Local	Coarse	0.999	0	98	197	100
DYHS	HZ	Local	Fine	1.000	0	100	201	103
DYHS	HZ	Local	Fine	0.999	0	99	199	101
DYHS	HZ	Mless	Coarse	1.000	0	131	263	135
DYHS	HZ	Mless	Coarse	0.999	0	91	183	95
DYHS	HZ	Mless	Fine	1.000	0	107	215	114
DYHS	HZ	Mless	Fine	0.999	0	98	197	103
DYHS	DS	LBFGS	—	1.000	0	210	423	212
DYHS	DS	Full	Coarse	1.000	0	108	244	110
DYHS	DS	Full	Coarse	0.999	0	124	273	126
DYHS	DS	Full	Fine	1.000	0	105	248	107
DYHS	DS	Full	Fine	0.999	0	103	235	105
DYHS	DS	Local	Coarse	1.000	0	104	216	106
DYHS	DS	Local	Coarse	0.999	0	100	211	102
DYHS	DS	Local	Fine	1.000	0	90	196	92
DYHS	DS	Local	Fine	0.999	0	82	178	84
DYHS	DS	Mless	Coarse	1.000	0	123	276	125
DYHS	DS	Mless	Coarse	0.999	0	149	337	152
DYHS	DS	Mless	Fine	1.000	0	110	252	112
DYHS	DS	Mless	Fine	0.999	0	112	260	114
HZ	HZ	LBFGS	—	1.000	0	153	307	154
HZ	HZ	Full	Coarse	1.000	0	113	227	118
HZ	HZ	Full	Coarse	0.999	0	103	207	105
HZ	HZ	Full	Fine	1.000	0	112	225	115
HZ	HZ	Full	Fine	0.999	0	92	185	94
HZ	HZ	Local	Coarse	1.000	0	91	183	93
HZ	HZ	Local	Coarse	0.999	0	96	193	98
HZ	HZ	Local	Fine	1.000	0	80	161	83
HZ	HZ	Local	Fine	0.999	0	79	159	82
HZ	HZ	Mless	Coarse	1.000	0	133	267	139
HZ	HZ	Mless	Coarse	0.999	0	110	221	113
HZ	HZ	Mless	Fine	1.000	0	102	205	109
HZ	HZ	Mless	Fine	0.999	0	106	213	111
HZ	DS	LBFGS	—	1.000	0	268	557	270
HZ	DS	Full	Coarse	1.000	0	98	232	100
HZ	DS	Full	Coarse	0.999	0	96	214	98
HZ	DS	Full	Fine	1.000	0	117	265	119
HZ	DS	Full	Fine	0.999	0	91	205	93
HZ	DS	Local	Coarse	1.000	0	98	214	100
HZ	DS	Local	Coarse	0.999	0	93	206	95
HZ	DS	Local	Fine	1.000	0	138	290	140
HZ	DS	Local	Fine	0.999	0	156	320	158
HZ	DS	Mless	Coarse	1.000	0	117	258	119
HZ	DS	Mless	Coarse	0.999	0	160	365	162
HZ	DS	Mless	Fine	1.000	0	203	441	205
HZ	DS	Mless	Fine	0.999	0	174	386	176

Table A.5 — *Results for problem* P3D *(level 5, 250,047 variables).*

β	linesearch	select	order	τ	status	iter	#f	#g
QN	HZ	LBFGS	—	1.000	0	1,131	2,244	3,375
QN	HZ	Full	Coarse	1.000	0	297	460	757
QN	HZ	Full	Coarse	0.999	0	302	467	769
QN	HZ	Full	Fine	1.000	0	214	338	552
QN	HZ	Full	Fine	0.999	0	208	321	529
QN	HZ	Local	Coarse	1.000	0	296	448	744
QN	HZ	Local	Coarse	0.999	0	304	468	772
QN	HZ	Local	Fine	1.000	0	272	433	705
QN	HZ	Local	Fine	0.999	0	317	509	826
QN	HZ	Mless	Coarse	1.000	0	243	360	603
QN	HZ	Mless	Coarse	0.999	0	135	189	324
QN	HZ	Mless	Fine	1.000	0	137	200	337
QN	HZ	Mless	Fine	0.999	0	144	212	356
QN	DS	LBFGS	—	1.000	0	921	945	922
QN	DS	Full	Coarse	1.000	0	219	280	220
QN	DS	Full	Coarse	0.999	0	274	333	275
QN	DS	Full	Fine	1.000	0	255	309	256
QN	DS	Full	Fine	0.999	0	255	309	256
QN	DS	Local	Coarse	1.000	0	255	317	256
QN	DS	Local	Coarse	0.999	0	266	324	267
QN	DS	Local	Fine	1.000	0	241	292	242
QN	DS	Local	Fine	0.999	0	272	336	273
QN	DS	Mless	Coarse	1.000	0	216	277	217
QN	DS	Mless	Coarse	0.999	0	171	216	172
QN	DS	Mless	Fine	1.000	0	169	222	170
QN	DS	Mless	Fine	0.999	0	181	236	185
DYHS	HZ	LBFGS	—	1.000	0	677	1,355	678
DYHS	HZ	Full	Coarse	1.000	0	277	555	283
DYHS	HZ	Full	Coarse	0.999	0	241	483	245
DYHS	HZ	Full	Fine	1.000	0	251	503	254
DYHS	HZ	Full	Fine	0.999	0	241	483	248
DYHS	HZ	Local	Coarse	1.000	0	306	613	312
DYHS	HZ	Local	Coarse	0.999	0	230	461	242
DYHS	HZ	Local	Fine	1.000	0	246	493	253
DYHS	HZ	Local	Fine	0.999	0	270	541	277
DYHS	HZ	Mless	Coarse	1.000	0	207	415	222
DYHS	HZ	Mless	Coarse	0.999	0	135	271	146
DYHS	HZ	Mless	Fine	1.000	0	149	299	166
DYHS	HZ	Mless	Fine	0.999	0	202	405	222
DYHS	DS	LBFGS	—	1.000	0	840	1,682	841
DYHS	DS	Full	Coarse	1.000	0	298	658	299
DYHS	DS	Full	Coarse	0.999	0	253	566	254
DYHS	DS	Full	Fine	1.000	0	246	529	247
DYHS	DS	Full	Fine	0.999	0	226	500	227
DYHS	DS	Local	Coarse	1.000	0	241	531	242
DYHS	DS	Local	Coarse	0.999	0	331	718	332
DYHS	DS	Local	Fine	1.000	0	301	681	303
DYHS	DS	Local	Fine	0.999	0	301	660	302
DYHS	DS	Mless	Coarse	1.000	0	235	544	238
DYHS	DS	Mless	Coarse	0.999	0	191	450	198
DYHS	DS	Mless	Fine	1.000	0	193	439	194
DYHS	DS	Mless	Fine	0.999	0	167	370	168
HZ	HZ	LBFGS	—	1.000	0	677	1,355	678
HZ	HZ	Full	Coarse	1.000	0	253	507	257
HZ	HZ	Full	Coarse	0.999	0	236	473	238
HZ	HZ	Full	Fine	1.000	0	217	435	219
HZ	HZ	Full	Fine	0.999	0	215	431	217
HZ	HZ	Local	Coarse	1.000	0	241	483	244
HZ	HZ	Local	Coarse	0.999	0	266	533	277
HZ	HZ	Local	Fine	1.000	0	260	521	266
HZ	HZ	Local	Fine	0.999	0	226	453	238
HZ	HZ	Mless	Coarse	1.000	0	190	381	200
HZ	HZ	Mless	Coarse	0.999	0	261	523	284
HZ	HZ	Mless	Fine	1.000	0	113	227	131
HZ	HZ	Mless	Fine	0.999	0	265	531	294
HZ	DS	LBFGS	—	1.000	0	1,220	2,519	1,222
HZ	DS	Full	Coarse	1.000	0	309	685	310
HZ	DS	Full	Coarse	0.999	0	302	688	303
HZ	DS	Full	Fine	1.000	0	222	504	223
HZ	DS	Full	Fine	0.999	0	371	795	372
HZ	DS	Local	Coarse	1.000	0	189	434	191
HZ	DS	Local	Coarse	0.999	0	216	478	217
HZ	DS	Local	Fine	1.000	0	353	760	354
HZ	DS	Local	Fine	0.999	0	323	693	326
HZ	DS	Mless	Coarse	1.000	0	334	731	335
HZ	DS	Mless	Coarse	0.999	0	280	623	281
HZ	DS	Mless	Fine	1.000	0	508	1,077	510
HZ	DS	Mless	Fine	0.999	0	269	597	274

Table A.6 — *Results for problem* DEPT *(level 8, 261,121 variables).*

β	linesearch	select	order	τ	status	iter	#f	#g
QN	HZ	LBFGS	—	1.000	0	1,699	3,395	5,094
QN	HZ	Full	Coarse	1.000	0	400	611	1,011
QN	HZ	Full	Coarse	0.999	0	396	609	1,005
QN	HZ	Full	Fine	1.000	0	554	864	1,418
QN	HZ	Full	Fine	0.999	0	415	653	1,068
QN	HZ	Local	Coarse	1.000	0	451	695	1,146
QN	HZ	Local	Coarse	0.999	0	613	947	1,560
QN	HZ	Local	Fine	1.000	0	383	609	992
QN	HZ	Local	Fine	0.999	0	520	826	1,346
QN	HZ	Mless	Coarse	1.000	0	347	527	874
QN	HZ	Mless	Coarse	0.999	0	321	489	810
QN	HZ	Mless	Fine	1.000	0	393	604	997
QN	HZ	Mless	Fine	0.999	0	249	379	628
QN	DS	LBFGS	—	1.000	0	1,840	1,842	1,841
QN	DS	Full	Coarse	1.000	0	423	506	425
QN	DS	Full	Coarse	0.999	0	694	809	695
QN	DS	Full	Fine	1.000	0	400	475	401
QN	DS	Full	Fine	0.999	0	612	713	613
QN	DS	Local	Coarse	1.000	0	602	724	603
QN	DS	Local	Coarse	0.999	0	457	548	458
QN	DS	Local	Fine	1.000	0	669	765	670
QN	DS	Local	Fine	0.999	0	510	598	511
QN	DS	Mless	Coarse	1.000	0	295	378	296
QN	DS	Mless	Coarse	0.999	0	510	607	511
QN	DS	Mless	Fine	1.000	0	475	596	476
QN	DS	Mless	Fine	0.999	0	409	492	410
DYHS	HZ	LBFGS	—	1.000	0	1,591	3,183	1,592
DYHS	HZ	Full	Coarse	1.000	0	477	955	485
DYHS	HZ	Full	Coarse	0.999	0	408	817	415
DYHS	HZ	Full	Fine	1.000	0	287	575	291
DYHS	HZ	Full	Fine	0.999	0	668	1,337	681
DYHS	HZ	Local	Coarse	1.000	0	359	719	370
DYHS	HZ	Local	Coarse	0.999	0	425	855	441
DYHS	HZ	Local	Fine	1.000	0	448	897	457
DYHS	HZ	Local	Fine	0.999	0	401	803	408
DYHS	HZ	Mless	Coarse	1.000	0	478	963	502
DYHS	HZ	Mless	Coarse	0.999	0	305	613	329
DYHS	HZ	Mless	Fine	1.000	0	353	710	386
DYHS	HZ	Mless	Fine	0.999	0	388	779	417
DYHS	DS	LBFGS	—	1.000	0	1,647	3,296	1,648
DYHS	DS	Full	Coarse	1.000	0	528	1,185	529
DYHS	DS	Full	Coarse	0.999	0	671	1,462	672
DYHS	DS	Full	Fine	1.000	0	501	1,105	502
DYHS	DS	Full	Fine	0.999	0	933	2,011	934
DYHS	DS	Local	Coarse	1.000	0	436	960	437
DYHS	DS	Local	Coarse	0.999	0	570	1,258	571
DYHS	DS	Local	Fine	1.000	0	738	1,603	739
DYHS	DS	Local	Fine	0.999	0	637	1,387	638
DYHS	DS	Mless	Coarse	1.000	0	432	994	436
DYHS	DS	Mless	Coarse	0.999	0	464	1,047	465
DYHS	DS	Mless	Fine	1.000	0	417	973	418
DYHS	DS	Mless	Fine	0.999	0	526	1,149	527
HZ	HZ	LBFGS	—	1.000	0	1,689	3,379	1,690
HZ	HZ	Full	Coarse	1.000	0	377	755	383
HZ	HZ	Full	Coarse	0.999	0	376	753	380
HZ	HZ	Full	Fine	1.000	0	396	793	403
HZ	HZ	Full	Fine	0.999	0	672	1,345	681
HZ	HZ	Local	Coarse	1.000	0	458	917	465
HZ	HZ	Local	Coarse	0.999	0	275	551	282
HZ	HZ	Local	Fine	1.000	0	513	1,027	525
HZ	HZ	Local	Fine	0.999	0	303	607	309
HZ	HZ	Mless	Coarse	1.000	0	269	543	284
HZ	HZ	Mless	Coarse	0.999	0	222	445	235
HZ	HZ	Mless	Fine	1.000	0	310	621	330
HZ	HZ	Mless	Fine	0.999	0	245	491	268
HZ	DS	LBFGS	—	1.000	0	2,434	4,934	2,435
HZ	DS	Full	Coarse	1.000	0	426	943	427
HZ	DS	Full	Coarse	0.999	0	586	1,269	587
HZ	DS	Full	Fine	1.000	0	656	1,368	660
HZ	DS	Full	Fine	0.999	0	515	1,104	517
HZ	DS	Local	Coarse	1.000	0	308	695	309
HZ	DS	Local	Coarse	0.999	0	500	1,091	501
HZ	DS	Local	Fine	1.000	0	875	1,811	878
HZ	DS	Local	Fine	0.999	0	526	1,126	528
HZ	DS	Mless	Coarse	1.000	0	502	1,115	503
HZ	DS	Mless	Coarse	0.999	0	356	776	358
HZ	DS	Mless	Fine	1.000	0	622	1,335	629
HZ	DS	Mless	Fine	0.999	0	238	525	240

Table A.7 — *Results for problem* DODC *(level 8, 261,121 variables).*

β	linesearch	select	order	τ	status	iter	#f	#g
QN	HZ	LBFGS	—	1.000	0	1,208	2,391	3,599
QN	HZ	Full	Coarse	1.000	0	823	1,304	2,127
QN	HZ	Full	Coarse	0.999	0	997	1,594	2,591
QN	HZ	Full	Fine	1.000	0	800	1,312	2,112
QN	HZ	Full	Fine	0.999	0	933	1,557	2,490
QN	HZ	Local	Coarse	1.000	0	771	1,219	1,990
QN	HZ	Local	Coarse	0.999	0	704	1,099	1,803
QN	HZ	Local	Fine	1.000	0	806	1,344	2,150
QN	HZ	Local	Fine	0.999	0	753	1,252	2,005
QN	HZ	Mless	Coarse	1.000	0	789	1,231	2,020
QN	HZ	Mless	Coarse	0.999	0	752	1,209	1,961
QN	HZ	Mless	Fine	1.000	0	665	1,089	1,754
QN	HZ	Mless	Fine	0.999	0	690	1,144	1,834
QN	DS	LBFGS	—	1.000	0	1,260	1,275	1,261
QN	DS	Full	Coarse	1.000	0	799	1,032	800
QN	DS	Full	Coarse	0.999	0	907	1,068	908
QN	DS	Full	Fine	1.000	0	789	926	790
QN	DS	Full	Fine	0.999	0	909	1,064	910
QN	DS	Local	Coarse	1.000	0	824	988	825
QN	DS	Local	Coarse	0.999	0	723	883	724
QN	DS	Local	Fine	1.000	0	819	946	820
QN	DS	Local	Fine	0.999	0	816	966	817
QN	DS	Mless	Coarse	1.000	0	812	1,037	813
QN	DS	Mless	Coarse	0.999	0	805	1,036	806
QN	DS	Mless	Fine	1.000	0	796	1,016	797
QN	DS	Mless	Fine	0.999	0	802	989	808
DYHS	HZ	LBFGS	—	1.000	0	1,241	2,490	1,249
DYHS	HZ	Full	Coarse	1.000	0	744	1,512	788
DYHS	HZ	Full	Coarse	0.999	0	957	1,926	987
DYHS	HZ	Full	Fine	1.000	0	809	1,630	836
DYHS	HZ	Full	Fine	0.999	0	953	1,910	963
DYHS	HZ	Local	Coarse	1.000	0	966	1,949	1,005
DYHS	HZ	Local	Coarse	0.999	0	800	1,630	854
DYHS	HZ	Local	Fine	1.000	0	501	1,007	516
DYHS	HZ	Local	Fine	0.999	0	594	1,195	618
DYHS	HZ	Mless	Coarse	1.000	0	749	1,534	819
DYHS	HZ	Mless	Coarse	0.999	0	731	1,509	826
DYHS	HZ	Mless	Fine	1.000	0	654	1,326	692
DYHS	HZ	Mless	Fine	0.999	0	748	1,529	839
DYHS	DS	LBFGS	—	1.000	0	1,251	2,504	1,252
DYHS	DS	Full	Coarse	1.000	0	854	1,929	855
DYHS	DS	Full	Coarse	0.999	0	814	1,773	815
DYHS	DS	Full	Fine	1.000	0	992	2,154	993
DYHS	DS	Full	Fine	0.999	0	1,101	2,378	1,102
DYHS	DS	Local	Coarse	1.000	0	775	1,690	776
DYHS	DS	Local	Coarse	0.999	0	793	1,745	794
DYHS	DS	Local	Fine	1.000	0	745	1,627	746
DYHS	DS	Local	Fine	0.999	0	831	1,817	832
DYHS	DS	Mless	Coarse	1.000	0	776	1,828	779
DYHS	DS	Mless	Coarse	0.999	0	859	1,979	861
DYHS	DS	Mless	Fine	1.000	0	821	1,886	823
DYHS	DS	Mless	Fine	0.999	0	818	1,840	824
HZ	HZ	LBFGS	—	1.000	0	1,081	2,163	1,082
HZ	HZ	Full	Coarse	1.000	0	826	1,668	859
HZ	HZ	Full	Coarse	0.999	0	951	1,910	975
HZ	HZ	Full	Fine	1.000	0	658	1,321	675
HZ	HZ	Full	Fine	0.999	0	791	1,588	811
HZ	HZ	Local	Coarse	1.000	0	652	1,315	674
HZ	HZ	Local	Coarse	0.999	0	552	1,122	590
HZ	HZ	Local	Fine	1.000	0	620	1,248	636
HZ	HZ	Local	Fine	0.999	0	642	1,293	662
HZ	HZ	Mless	Coarse	1.000	0	782	1,579	818
HZ	HZ	Mless	Coarse	0.999	0	831	1,697	907
HZ	HZ	Mless	Fine	1.000	0	638	1,289	673
HZ	HZ	Mless	Fine	0.999	0	850	1,746	965
HZ	DS	LBFGS	—	1.000	0	1,292	2,653	1,295
HZ	DS	Full	Coarse	1.000	0	836	1,893	837
HZ	DS	Full	Coarse	0.999	0	949	2,078	950
HZ	DS	Full	Fine	1.000	0	888	1,939	889
HZ	DS	Full	Fine	0.999	0	1,033	2,290	1,035
HZ	DS	Local	Coarse	1.000	0	710	1,613	711
HZ	DS	Local	Coarse	0.999	0	764	1,684	765
HZ	DS	Local	Fine	1.000	0	896	1,950	903
HZ	DS	Local	Fine	0.999	0	866	1,900	869
HZ	DS	Mless	Coarse	1.000	0	1,087	2,454	1,088
HZ	DS	Mless	Coarse	0.999	0	1,225	2,683	1,232
HZ	DS	Mless	Fine	1.000	0	1,060	2,341	1,064
HZ	DS	Mless	Fine	0.999	0	1,214	2,659	1,236

Table A.8 — *Results for problem* `MINS-SB` *(level 8, 261,121 variables).*

β	linesearch	select	order	τ	status	iter	#f	#g
QN	HZ	LBFGS	—	1.000	0	1,991	3,976	5,967
QN	HZ	Full	Coarse	1.000	0	3,932	6,494	10,426
QN	HZ	Full	Coarse	0.999	0	3,773	6,225	9,998
QN	HZ	Full	Fine	1.000	0	4,383	7,804	12,187
QN	HZ	Full	Fine	0.999	0	4,565	8,063	12,628
QN	HZ	Local	Coarse	1.000	0	4,758	7,981	12,739
QN	HZ	Local	Coarse	0.999	0	4,324	7,199	11,523
QN	HZ	Local	Fine	1.000	0	3,273	5,880	9,153
QN	HZ	Local	Fine	0.999	0	3,334	5,929	9,263
QN	HZ	Mless	Coarse	1.000	0	4,179	6,903	11,082
QN	HZ	Mless	Coarse	0.999	0	4,200	6,909	11,109
QN	HZ	Mless	Fine	1.000	0	3,683	6,352	10,035
QN	HZ	Mless	Fine	0.999	0	3,758	6,391	10,149
QN	DS	LBFGS	—	1.000	0	2,107	2,143	2,108
QN	DS	Full	Coarse	1.000	0	3,392	4,005	3,393
QN	DS	Full	Coarse	0.999	0	3,917	4,596	3,918
QN	DS	Full	Fine	1.000	0	4,151	4,688	4,152
QN	DS	Full	Fine	0.999	0	4,628	5,253	4,629
QN	DS	Local	Coarse	1.000	0	4,254	4,911	4,255
QN	DS	Local	Coarse	0.999	0	4,342	4,987	4,344
QN	DS	Local	Fine	1.000	0	3,834	4,382	3,835
QN	DS	Local	Fine	0.999	0	3,874	4,405	3,875
QN	DS	Mless	Coarse	1.000	0	4,476	5,527	4,482
QN	DS	Mless	Coarse	0.999	0	4,771	5,880	4,794
QN	DS	Mless	Fine	1.000	0	4,337	5,156	4,339
QN	DS	Mless	Fine	0.999	0	4,352	5,204	4,368
DYHS	HZ	LBFGS	—	1.000	0	1,960	3,921	1,961
DYHS	HZ	Full	Coarse	1.000	0	3,724	7,461	3,779
DYHS	HZ	Full	Coarse	0.999	0	4,028	8,069	4,085
DYHS	HZ	Full	Fine	1.000	0	4,872	9,756	4,919
DYHS	HZ	Full	Fine	0.999	0	4,691	9,394	4,740
DYHS	HZ	Local	Coarse	1.000	0	4,860	9,725	4,923
DYHS	HZ	Local	Coarse	0.999	0	4,472	8,957	4,535
DYHS	HZ	Local	Fine	1.000	0	3,386	6,779	3,415
DYHS	HZ	Local	Fine	0.999	0	3,135	6,279	3,168
DYHS	HZ	Mless	Coarse	1.000	0	4,252	8,532	4,348
DYHS	HZ	Mless	Coarse	0.999	0	4,334	8,698	4,481
DYHS	HZ	Mless	Fine	1.000	0	3,581	7,188	3,671
DYHS	HZ	Mless	Fine	0.999	0	3,443	6,965	3,740
DYHS	DS	LBFGS	—	1.000	0	2,083	4,168	2,084
DYHS	DS	Full	Coarse	1.000	0	3,510	7,680	3,511
DYHS	DS	Full	Coarse	0.999	0	4,176	9,012	4,177
DYHS	DS	Full	Fine	1.000	0	4,881	10,276	4,882
DYHS	DS	Full	Fine	0.999	0	4,883	10,388	4,884
DYHS	DS	Local	Coarse	1.000	0	5,107	10,875	5,108
DYHS	DS	Local	Coarse	0.999	0	4,678	10,086	4,681
DYHS	DS	Local	Fine	1.000	0	4,097	8,698	4,100
DYHS	DS	Local	Fine	0.999	0	4,341	9,223	4,342
DYHS	DS	Mless	Coarse	1.000	0	4,931	11,171	4,951
DYHS	DS	Mless	Coarse	0.999	0	4,547	10,137	4,565
DYHS	DS	Mless	Fine	1.000	0	4,229	9,306	4,232
DYHS	DS	Mless	Fine	0.999	0	4,151	9,180	4,162
HZ	HZ	LBFGS	—	1.000	0	1,950	3,901	1,951
HZ	HZ	Full	Coarse	1.000	0	3,917	7,852	3,973
HZ	HZ	Full	Coarse	0.999	0	4,086	8,182	4,131
HZ	HZ	Full	Fine	1.000	0	4,913	9,841	4,961
HZ	HZ	Full	Fine	0.999	0	4,367	8,741	4,409
HZ	HZ	Local	Coarse	1.000	0	5,092	10,201	5,165
HZ	HZ	Local	Coarse	0.999	0	4,475	8,966	4,536
HZ	HZ	Local	Fine	1.000	0	3,287	6,580	3,307
HZ	HZ	Local	Fine	0.999	0	3,366	6,739	3,398
HZ	HZ	Mless	Coarse	1.000	0	4,217	8,459	4,313
HZ	HZ	Mless	Coarse	0.999	0	4,414	8,875	4,627
HZ	HZ	Mless	Fine	1.000	0	3,392	6,823	3,503
HZ	HZ	Mless	Fine	0.999	0	3,461	6,972	3,660
HZ	DS	LBFGS	—	1.000	0	2,712	5,563	2,715
HZ	DS	Full	Coarse	1.000	0	4,143	8,995	4,144
HZ	DS	Full	Coarse	0.999	0	4,268	9,141	4,272
HZ	DS	Full	Fine	1.000	0	4,438	9,504	4,440
HZ	DS	Full	Fine	0.999	0	4,465	9,554	4,471
HZ	DS	Local	Coarse	1.000	0	4,094	8,813	4,095
HZ	DS	Local	Coarse	0.999	0	4,920	10,604	4,923
HZ	DS	Local	Fine	1.000	0	4,969	10,625	4,989
HZ	DS	Local	Fine	0.999	0	5,113	10,890	5,131
HZ	DS	Mless	Coarse	1.000	0	5,973	13,180	5,979
HZ	DS	Mless	Coarse	0.999	0	7,114	15,568	7,162
HZ	DS	Mless	Fine	1.000	0	6,880	14,889	6,908
HZ	DS	Mless	Fine	0.999	0	7,820	16,836	7,905

Table A.9 — *Results for problem* `MINS-OB` *(level 8, 261,121 variables).*

β	linesearch	select	order	τ	status	iter	#f	#g
QN	HZ	LBFGS	—	1.000	0	1,499	2,990	4,489
QN	HZ	Full	Coarse	1.000	0	1,720	2,747	4,467
QN	HZ	Full	Coarse	0.999	0	1,702	2,730	4,432
QN	HZ	Full	Fine	1.000	0	1,624	2,686	4,310
QN	HZ	Full	Fine	0.999	0	1,781	3,020	4,801
QN	HZ	Local	Coarse	1.000	0	1,700	2,727	4,427
QN	HZ	Local	Coarse	0.999	0	1,609	2,581	4,190
QN	HZ	Local	Fine	1.000	0	1,210	2,046	3,256
QN	HZ	Local	Fine	0.999	0	1,307	2,224	3,531
QN	HZ	Mless	Coarse	1.000	0	1,706	2,728	4,434
QN	HZ	Mless	Coarse	0.999	0	1,580	2,543	4,123
QN	HZ	Mless	Fine	1.000	0	1,401	2,310	3,711
QN	HZ	Mless	Fine	0.999	0	1,633	2,760	4,393
QN	DS	LBFGS	—	1.000	0	1,329	1,350	1,330
QN	DS	Full	Coarse	1.000	0	1,313	1,613	1,314
QN	DS	Full	Coarse	0.999	0	1,660	1,950	1,661
QN	DS	Full	Fine	1.000	0	1,601	1,933	1,602
QN	DS	Full	Fine	0.999	0	1,985	2,308	1,986
QN	DS	Local	Coarse	1.000	0	1,669	1,976	1,670
QN	DS	Local	Coarse	0.999	0	1,616	1,955	1,617
QN	DS	Local	Fine	1.000	0	1,614	1,938	1,615
QN	DS	Local	Fine	0.999	0	1,671	1,938	1,672
QN	DS	Mless	Coarse	1.000	0	1,553	1,983	1,555
QN	DS	Mless	Coarse	0.999	0	1,716	2,155	1,719
QN	DS	Mless	Fine	1.000	0	1,629	2,020	1,630
QN	DS	Mless	Fine	0.999	0	1,563	1,904	1,566
DYHS	HZ	LBFGS	—	1.000	0	1,227	2,455	1,228
DYHS	HZ	Full	Coarse	1.000	0	1,503	3,032	1,556
DYHS	HZ	Full	Coarse	0.999	0	1,832	3,679	1,880
DYHS	HZ	Full	Fine	1.000	0	1,435	2,873	1,453
DYHS	HZ	Full	Fine	0.999	0	1,825	3,655	1,845
DYHS	HZ	Local	Coarse	1.000	0	1,811	3,637	1,857
DYHS	HZ	Local	Coarse	0.999	0	1,496	3,008	1,544
DYHS	HZ	Local	Fine	1.000	0	1,203	2,415	1,237
DYHS	HZ	Local	Fine	0.999	0	1,132	2,269	1,156
DYHS	HZ	Mless	Coarse	1.000	0	1,701	3,443	1,797
DYHS	HZ	Mless	Coarse	0.999	0	1,662	3,393	1,825
DYHS	HZ	Mless	Fine	1.000	0	1,310	2,656	1,394
DYHS	HZ	Mless	Fine	0.999	0	1,518	3,121	1,721
DYHS	DS	LBFGS	—	1.000	0	1,591	3,184	1,592
DYHS	DS	Full	Coarse	1.000	0	1,737	3,877	1,738
DYHS	DS	Full	Coarse	0.999	0	1,991	4,362	1,992
DYHS	DS	Full	Fine	1.000	0	1,932	4,142	1,933
DYHS	DS	Full	Fine	0.999	0	1,958	4,248	1,959
DYHS	DS	Local	Coarse	1.000	0	1,774	3,830	1,775
DYHS	DS	Local	Coarse	0.999	0	1,737	3,790	1,738
DYHS	DS	Local	Fine	1.000	0	1,722	3,707	1,723
DYHS	DS	Local	Fine	0.999	0	1,470	3,182	1,471
DYHS	DS	Mless	Coarse	1.000	0	1,733	4,067	1,739
DYHS	DS	Mless	Coarse	0.999	0	1,793	4,135	1,805
DYHS	DS	Mless	Fine	1.000	0	1,769	4,002	1,774
DYHS	DS	Mless	Fine	0.999	0	1,716	3,897	1,729
HZ	HZ	LBFGS	—	1.000	0	1,337	2,676	1,339
HZ	HZ	Full	Coarse	1.000	0	1,620	3,269	1,682
HZ	HZ	Full	Coarse	0.999	0	1,702	3,413	1,732
HZ	HZ	Full	Fine	1.000	0	1,680	3,372	1,714
HZ	HZ	Full	Fine	0.999	0	1,518	3,045	1,544
HZ	HZ	Local	Coarse	1.000	0	1,357	2,736	1,415
HZ	HZ	Local	Coarse	0.999	0	1,510	3,035	1,557
HZ	HZ	Local	Fine	1.000	0	1,224	2,462	1,260
HZ	HZ	Local	Fine	0.999	0	1,219	2,452	1,256
HZ	HZ	Mless	Coarse	1.000	0	1,638	3,307	1,716
HZ	HZ	Mless	Coarse	0.999	0	1,601	3,249	1,713
HZ	HZ	Mless	Fine	1.000	0	1,361	2,742	1,419
HZ	HZ	Mless	Fine	0.999	0	1,443	2,949	1,601
HZ	DS	LBFGS	—	1.000	0	1,528	3,123	1,531
HZ	DS	Full	Coarse	1.000	0	1,687	3,737	1,688
HZ	DS	Full	Coarse	0.999	0	1,896	4,163	1,897
HZ	DS	Full	Fine	1.000	0	1,827	3,977	1,829
HZ	DS	Full	Fine	0.999	0	2,088	4,536	2,091
HZ	DS	Local	Coarse	1.000	0	1,741	3,755	1,742
HZ	DS	Local	Coarse	0.999	0	1,581	3,480	1,584
HZ	DS	Local	Fine	1.000	0	1,783	3,838	1,792
HZ	DS	Local	Fine	0.999	0	1,587	3,419	1,595
HZ	DS	Mless	Coarse	1.000	0	2,062	4,641	2,063
HZ	DS	Mless	Coarse	0.999	0	2,406	5,345	2,425
HZ	DS	Mless	Fine	1.000	0	2,117	4,700	2,130
HZ	DS	Mless	Fine	0.999	0	2,630	5,719	2,658

Table A.10 — *Results for problem* `MINS-DMSA` *(level 8, 261,121 variables).*

β	linesearch	select	order	τ	status	iter	#f	#g
QN	HZ	LBFGS	—	1.000	0	6,515	13,022	19,537
QN	HZ	Full	Coarse	1.000	0	4,759	5,933	10,692
QN	HZ	Full	Coarse	0.999	0	6,171	8,649	14,820
QN	HZ	Full	Fine	1.000	8	8,314	10,640	18,954
QN	HZ	Full	Fine	0.999	0	10,751	15,170	25,921
QN	HZ	Local	Coarse	1.000	0	3,926	5,190	9,116
QN	HZ	Local	Coarse	0.999	0	1,380	2,189	3,569
QN	HZ	Local	Fine	1.000	8	2,663	3,566	6,229
QN	HZ	Local	Fine	0.999	0	2,740	4,498	7,238
QN	HZ	Mless	Coarse	1.000	0	1,507	2,093	3,600
QN	HZ	Mless	Coarse	0.999	0	7,496	14,260	21,756
QN	HZ	Mless	Fine	1.000	0	3,440	4,790	8,230
QN	HZ	Mless	Fine	0.999	0	6,746	12,845	19,591
QN	DS	LBFGS	—	1.000	0	7,142	7,309	7,143
QN	DS	Full	Coarse	1.000	0	2,648	4,200	2,649
QN	DS	Full	Coarse	0.999	0	4,756	6,436	4,757
QN	DS	Full	Fine	1.000	0	7,494	11,150	7,496
QN	DS	Full	Fine	0.999	0	10,773	14,305	10,774
QN	DS	Local	Coarse	1.000	0	3,295	4,663	3,296
QN	DS	Local	Coarse	0.999	0	1,487	1,835	1,488
QN	DS	Local	Fine	1.000	0	4,653	6,235	4,654
QN	DS	Local	Fine	0.999	0	3,349	3,990	3,350
QN	DS	Mless	Coarse	1.000	0	1,757	2,592	1,761
QN	DS	Mless	Coarse	0.999	0	2,566	3,012	2,580
QN	DS	Mless	Fine	1.000	0	2,453	3,345	2,454
QN	DS	Mless	Fine	0.999	0	2,275	2,614	2,282
DYHS	HZ	LBFGS	—	1.000	0	6,471	12,943	6,472
DYHS	HZ	Full	Coarse	1.000	8	976	2,162	1,315
DYHS	HZ	Full	Coarse	0.999	0	5,707	11,612	6,358
DYHS	HZ	Full	Fine	1.000	8	3,550	7,326	4,260
DYHS	HZ	Full	Fine	0.999	0	8,763	17,715	9,627
DYHS	HZ	Local	Coarse	1.000	8	714	1,628	998
DYHS	HZ	Local	Coarse	0.999	0	1,868	3,807	2,072
DYHS	HZ	Local	Fine	1.000	0	5,552	11,206	6,077
DYHS	HZ	Local	Fine	0.999	0	2,931	5,897	3,099
DYHS	HZ	Mless	Coarse	1.000	8	1,360	2,860	1,617
DYHS	HZ	Mless	Coarse	0.999	0	11,206	22,426	11,545
DYHS	HZ	Mless	Fine	1.000	0	2,514	5,069	2,740
DYHS	HZ	Mless	Fine	0.999	0	6,592	13,218	6,856
DYHS	DS	LBFGS	—	1.000	0	7,822	15,664	7,823
DYHS	DS	Full	Coarse	1.000	0	3,115	8,073	3,116
DYHS	DS	Full	Coarse	0.999	0	4,797	11,355	4,798
DYHS	DS	Full	Fine	1.000	0	8,206	20,663	8,207
DYHS	DS	Full	Fine	0.999	0	9,501	22,249	9,502
DYHS	DS	Local	Coarse	1.000	0	2,836	6,831	2,837
DYHS	DS	Local	Coarse	0.999	0	1,781	3,988	1,782
DYHS	DS	Local	Fine	1.000	0	5,126	11,995	5,127
DYHS	DS	Local	Fine	0.999	0	1,737	3,773	1,738
DYHS	DS	Mless	Coarse	1.000	0	2,537	6,241	2,542
DYHS	DS	Mless	Coarse	0.999	0	2,970	6,344	2,982
DYHS	DS	Mless	Fine	1.000	0	3,472	8,311	3,474
DYHS	DS	Mless	Fine	0.999	0	3,166	6,814	3,174
HZ	HZ	LBFGS	—	1.000	0	6,462	12,925	6,463
HZ	HZ	Full	Coarse	1.000	8	499	1,156	724
HZ	HZ	Full	Coarse	0.999	8	5,015	10,234	5,599
HZ	HZ	Full	Fine	1.000	8	1,561	3,375	2,003
HZ	HZ	Full	Fine	0.999	0	7,390	14,953	8,126
HZ	HZ	Local	Coarse	1.000	0	3,290	6,724	3,775
HZ	HZ	Local	Coarse	0.999	0	1,918	3,859	2,066
HZ	HZ	Local	Fine	1.000	0	4,922	9,944	5,393
HZ	HZ	Local	Fine	0.999	0	2,461	4,965	2,617
HZ	HZ	Mless	Coarse	1.000	8	76	213	143
HZ	HZ	Mless	Coarse	0.999	0	6,355	12,745	6,601
HZ	HZ	Mless	Fine	1.000	0	3,092	6,257	3,370
HZ	HZ	Mless	Fine	0.999	0	6,561	13,127	6,765
HZ	DS	LBFGS	—	1.000	0	6,639	13,635	6,651
HZ	DS	Full	Coarse	1.000	0	3,807	9,737	3,808
HZ	DS	Full	Coarse	0.999	0	5,234	11,883	5,239
HZ	DS	Full	Fine	1.000	0	7,318	17,302	7,320
HZ	DS	Full	Fine	0.999	0	12,040	27,087	12,054
HZ	DS	Local	Coarse	1.000	0	3,266	7,798	3,267
HZ	DS	Local	Coarse	0.999	0	1,963	4,269	1,971
HZ	DS	Local	Fine	1.000	0	3,865	8,672	3,871
HZ	DS	Local	Fine	0.999	0	2,126	4,574	2,129
HZ	DS	Mless	Coarse	1.000	0	2,028	4,796	2,033
HZ	DS	Mless	Coarse	0.999	0	6,571	13,624	6,644
HZ	DS	Mless	Fine	1.000	0	4,065	8,900	4,079
HZ	DS	Mless	Fine	0.999	0	2,931	6,297	2,991

Table A.11 — *Results for problem* `IGNISC` *(level 8, 261,121 variables).*

β	linesearch	select	order	τ	status	iter	#f	#g
QN	HZ	LBFGS	—	1.000	0	1,051	2,101	3,152
QN	HZ	Full	Coarse	1.000	0	268	420	688
QN	HZ	Full	Coarse	0.999	0	247	376	623
QN	HZ	Full	Fine	1.000	0	198	314	512
QN	HZ	Full	Fine	0.999	0	191	296	487
QN	HZ	Local	Coarse	1.000	0	282	442	724
QN	HZ	Local	Coarse	0.999	0	268	408	676
QN	HZ	Local	Fine	1.000	0	241	378	619
QN	HZ	Local	Fine	0.999	0	214	351	565
QN	HZ	Mless	Coarse	1.000	0	279	421	700
QN	HZ	Mless	Coarse	0.999	0	430	741	1,171
QN	HZ	Mless	Fine	1.000	0	185	280	465
QN	HZ	Mless	Fine	0.999	0	143	213	356
QN	DS	LBFGS	—	1.000	0	871	894	872
QN	DS	Full	Coarse	1.000	0	210	276	211
QN	DS	Full	Coarse	0.999	0	301	365	302
QN	DS	Full	Fine	1.000	0	209	250	210
QN	DS	Full	Fine	0.999	0	206	245	207
QN	DS	Local	Coarse	1.000	0	240	299	241
QN	DS	Local	Coarse	0.999	0	192	245	193
QN	DS	Local	Fine	1.000	−7	6	30	25
QN	DS	Local	Fine	0.999	−7	6	31	26
QN	DS	Mless	Coarse	1.000	0	141	185	142
QN	DS	Mless	Coarse	0.999	0	156	202	157
QN	DS	Mless	Fine	1.000	0	157	205	158
QN	DS	Mless	Fine	0.999	0	197	245	199
DYHS	HZ	LBFGS	—	1.000	0	697	1,395	698
DYHS	HZ	Full	Coarse	1.000	0	304	609	315
DYHS	HZ	Full	Coarse	0.999	0	224	449	228
DYHS	HZ	Full	Fine	1.000	0	244	490	249
DYHS	HZ	Full	Fine	0.999	0	280	561	288
DYHS	HZ	Local	Coarse	1.000	0	243	487	247
DYHS	HZ	Local	Coarse	0.999	0	234	469	242
DYHS	HZ	Local	Fine	1.000	0	259	519	266
DYHS	HZ	Local	Fine	0.999	0	225	451	230
DYHS	HZ	Mless	Coarse	1.000	0	228	457	237
DYHS	HZ	Mless	Coarse	0.999	0	256	513	283
DYHS	HZ	Mless	Fine	1.000	0	158	317	168
DYHS	HZ	Mless	Fine	0.999	0	182	365	208
DYHS	DS	LBFGS	—	1.000	0	1,021	2,044	1,022
DYHS	DS	Full	Coarse	1.000	0	285	630	286
DYHS	DS	Full	Coarse	0.999	0	256	562	257
DYHS	DS	Full	Fine	1.000	0	228	512	229
DYHS	DS	Full	Fine	0.999	0	208	452	209
DYHS	DS	Local	Coarse	1.000	0	308	695	309
DYHS	DS	Local	Coarse	0.999	0	269	609	270
DYHS	DS	Local	Fine	1.000	−7	6	36	25
DYHS	DS	Local	Fine	0.999	−7	6	37	26
DYHS	DS	Mless	Coarse	1.000	0	206	471	207
DYHS	DS	Mless	Coarse	0.999	0	233	526	234
DYHS	DS	Mless	Fine	1.000	0	190	429	191
DYHS	DS	Mless	Fine	0.999	0	340	756	344
HZ	HZ	LBFGS	—	1.000	0	697	1,395	698
HZ	HZ	Full	Coarse	1.000	0	249	499	253
HZ	HZ	Full	Coarse	0.999	0	208	417	212
HZ	HZ	Full	Fine	1.000	0	268	537	271
HZ	HZ	Full	Fine	0.999	0	232	466	238
HZ	HZ	Local	Coarse	1.000	0	311	623	317
HZ	HZ	Local	Coarse	0.999	0	218	437	225
HZ	HZ	Local	Fine	1.000	0	239	479	248
HZ	HZ	Local	Fine	0.999	0	197	395	201
HZ	HZ	Mless	Coarse	1.000	0	221	443	227
HZ	HZ	Mless	Coarse	0.999	0	262	525	281
HZ	HZ	Mless	Fine	1.000	0	163	327	174
HZ	HZ	Mless	Fine	0.999	0	134	269	157
HZ	DS	LBFGS	—	1.000	0	1,376	2,839	1,384
HZ	DS	Full	Coarse	1.000	0	195	434	196
HZ	DS	Full	Coarse	0.999	0	325	723	326
HZ	DS	Full	Fine	1.000	−7	66	169	85
HZ	DS	Full	Fine	0.999	0	220	476	221
HZ	DS	Local	Coarse	1.000	0	281	615	282
HZ	DS	Local	Coarse	0.999	0	213	480	214
HZ	DS	Local	Fine	1.000	−7	6	36	25
HZ	DS	Local	Fine	0.999	−7	6	36	25
HZ	DS	Mless	Coarse	1.000	0	232	509	233
HZ	DS	Mless	Coarse	0.999	0	242	531	246
HZ	DS	Mless	Fine	1.000	0	272	604	274
HZ	DS	Mless	Fine	0.999	0	526	1,101	533

Table A.12 — *Results for problem* `DSSC` *(level 8, 261,121 variables).*

β	linesearch	select	order	τ	status	iter	#f	#g
QN	HZ	LBFGS	—	1.000	0	1,361	2,696	4,057
QN	HZ	Full	Coarse	1.000	0	325	494	819
QN	HZ	Full	Coarse	0.999	0	293	448	741
QN	HZ	Full	Fine	1.000	0	282	439	721
QN	HZ	Full	Fine	0.999	0	289	451	740
QN	HZ	Local	Coarse	1.000	0	366	570	936
QN	HZ	Local	Coarse	0.999	0	281	432	713
QN	HZ	Local	Fine	1.000	0	290	464	754
QN	HZ	Local	Fine	0.999	0	307	485	792
QN	HZ	Mless	Coarse	1.000	0	239	350	589
QN	HZ	Mless	Coarse	0.999	0	184	266	450
QN	HZ	Mless	Fine	1.000	0	181	267	448
QN	HZ	Mless	Fine	0.999	0	285	466	751
QN	DS	LBFGS	—	1.000	0	1,049	1,071	1,050
QN	DS	Full	Coarse	1.000	0	221	281	222
QN	DS	Full	Coarse	0.999	0	259	322	260
QN	DS	Full	Fine	1.000	0	254	306	255
QN	DS	Full	Fine	0.999	0	302	354	303
QN	DS	Local	Coarse	1.000	0	232	291	233
QN	DS	Local	Coarse	0.999	0	243	295	244
QN	DS	Local	Fine	1.000	0	292	354	293
QN	DS	Local	Fine	0.999	0	252	308	253
QN	DS	Mless	Coarse	1.000	0	259	336	260
QN	DS	Mless	Coarse	0.999	0	223	282	224
QN	DS	Mless	Fine	1.000	0	167	215	168
QN	DS	Mless	Fine	0.999	0	176	225	179
DYHS	HZ	LBFGS	—	1.000	0	896	1,793	897
DYHS	HZ	Full	Coarse	1.000	0	279	559	281
DYHS	HZ	Full	Coarse	0.999	0	214	429	219
DYHS	HZ	Full	Fine	1.000	0	226	453	231
DYHS	HZ	Full	Fine	0.999	0	259	519	261
DYHS	HZ	Local	Coarse	1.000	0	330	661	336
DYHS	HZ	Local	Coarse	0.999	0	304	609	312
DYHS	HZ	Local	Fine	1.000	0	313	627	319
DYHS	HZ	Local	Fine	0.999	0	280	561	292
DYHS	HZ	Mless	Coarse	1.000	0	235	471	246
DYHS	HZ	Mless	Coarse	0.999	0	191	383	206
DYHS	HZ	Mless	Fine	1.000	0	192	385	202
DYHS	HZ	Mless	Fine	0.999	0	193	387	228
DYHS	DS	LBFGS	—	1.000	0	1,007	2,016	1,008
DYHS	DS	Full	Coarse	1.000	0	282	630	283
DYHS	DS	Full	Coarse	0.999	0	263	584	264
DYHS	DS	Full	Fine	1.000	0	210	460	211
DYHS	DS	Full	Fine	0.999	0	279	607	280
DYHS	DS	Local	Coarse	1.000	0	324	715	325
DYHS	DS	Local	Coarse	0.999	0	317	692	318
DYHS	DS	Local	Fine	1.000	0	285	621	286
DYHS	DS	Local	Fine	0.999	0	289	617	291
DYHS	DS	Mless	Coarse	1.000	0	261	592	263
DYHS	DS	Mless	Coarse	0.999	0	246	571	250
DYHS	DS	Mless	Fine	1.000	0	153	359	154
DYHS	DS	Mless	Fine	0.999	0	191	434	194
HZ	HZ	LBFGS	—	1.000	0	896	1,793	897
HZ	HZ	Full	Coarse	1.000	0	300	624	326
HZ	HZ	Full	Coarse	0.999	0	272	545	274
HZ	HZ	Full	Fine	1.000	0	240	481	245
HZ	HZ	Full	Fine	0.999	0	272	545	277
HZ	HZ	Local	Coarse	1.000	0	318	637	326
HZ	HZ	Local	Coarse	0.999	0	304	609	313
HZ	HZ	Local	Fine	1.000	0	244	489	253
HZ	HZ	Local	Fine	0.999	0	320	641	328
HZ	HZ	Mless	Coarse	1.000	0	248	497	253
HZ	HZ	Mless	Coarse	0.999	0	221	443	236
HZ	HZ	Mless	Fine	1.000	0	227	455	274
HZ	HZ	Mless	Fine	0.999	0	214	429	247
HZ	DS	LBFGS	—	1.000	0	1,050	2,165	1,052
HZ	DS	Full	Coarse	1.000	0	291	655	292
HZ	DS	Full	Coarse	0.999	0	434	944	435
HZ	DS	Full	Fine	1.000	0	281	615	282
HZ	DS	Full	Fine	0.999	0	264	585	265
HZ	DS	Local	Coarse	1.000	0	249	552	250
HZ	DS	Local	Coarse	0.999	0	263	596	264
HZ	DS	Local	Fine	1.000	0	214	473	218
HZ	DS	Local	Fine	0.999	0	501	1,046	504
HZ	DS	Mless	Coarse	1.000	0	264	599	265
HZ	DS	Mless	Coarse	0.999	0	4,257	8,560	4,263
HZ	DS	Mless	Fine	1.000	0	326	716	327
HZ	DS	Mless	Fine	0.999	0	338	729	341

Table A.13 — *Results for problem* `BRATU` *(level 8, 261,121 variables).*

β	linesearch	select	order	τ	status	iter	#f	#g
QN	HZ	LBFGS	—	1.000	0	28,933	57,848	86,781
QN	HZ	Full	Coarse	1.000	0	30,063	34,683	64,746
QN	HZ	Full	Coarse	0.999	0	20,870	26,531	47,401
QN	HZ	Full	Fine	1.000	0	22,759	25,785	48,544
QN	HZ	Full	Fine	0.999	0	21,467	28,736	50,203
QN	HZ	Local	Coarse	1.000	0	11,594	13,580	25,174
QN	HZ	Local	Coarse	0.999	0	40,262	79,306	119,568
QN	HZ	Local	Fine	1.000	0	11,681	14,198	25,879
QN	HZ	Local	Fine	0.999	0	65,863	130,588	196,451
QN	HZ	Mless	Coarse	1.000	0	14,422	18,488	32,910
QN	HZ	Mless	Coarse	0.999	0	460,429	919,338	344,941
QN	HZ	Mless	Fine	1.000	0	15,060	19,934	34,994
QN	HZ	Mless	Fine	0.999	0	423,963	846,496	317,614
QN	DS	LBFGS	—	1.000	12	130,051	133,142	130,051
QN	DS	Full	Coarse	1.000	0	42,006	67,802	42,007
QN	DS	Full	Coarse	0.999	0	24,142	34,132	24,143
QN	DS	Full	Fine	1.000	0	35,948	57,713	35,949
QN	DS	Full	Fine	0.999	12	21,063	29,400	21,063
QN	DS	Local	Coarse	1.000	0	9,776	14,318	9,777
QN	DS	Local	Coarse	0.999	0	5,697	6,790	5,698
QN	DS	Local	Fine	1.000	0	10,629	14,662	10,630
QN	DS	Local	Fine	0.999	0	5,718	6,707	5,719
QN	DS	Mless	Coarse	1.000	0	13,101	19,192	13,109
QN	DS	Mless	Coarse	0.999	0	47,200	52,292	47,363
QN	DS	Mless	Fine	1.000	0	16,718	22,928	16,726
QN	DS	Mless	Fine	0.999	0	19,743	22,219	19,776
DYHS	HZ	LBFGS	—	1.000	0	20,449	40,677	20,674
DYHS	HZ	Full	Coarse	1.000	0	33,209	67,698	40,262
DYHS	HZ	Full	Coarse	0.999	0	20,271	40,715	23,393
DYHS	HZ	Full	Fine	1.000	0	29,485	60,738	37,379
DYHS	HZ	Full	Fine	0.999	0	25,171	50,532	29,772
DYHS	HZ	Local	Coarse	1.000	0	12,777	26,110	15,336
DYHS	HZ	Local	Coarse	0.999	0	123,395	246,671	123,782
DYHS	HZ	Local	Fine	1.000	0	11,639	23,420	13,274
DYHS	HZ	Local	Fine	0.999	0	69,692	139,206	70,166
DYHS	HZ	Mless	Coarse	1.000	0	13,272	26,827	15,141
DYHS	HZ	Mless	Coarse	0.999	0	444,361	885,886	447,771
DYHS	HZ	Mless	Fine	1.000	0	13,754	27,654	15,212
DYHS	HZ	Mless	Fine	0.999	0	439,886	878,698	441,514
DYHS	DS	LBFGS	—	1.000	12	130,051	260,335	130,051
DYHS	DS	Full	Coarse	1.000	0	47,969	125,882	47,971
DYHS	DS	Full	Coarse	0.999	0	36,051	75,684	36,052
DYHS	DS	Full	Fine	1.000	0	27,314	71,534	27,315
DYHS	DS	Full	Fine	0.999	0	56,895	119,970	56,896
DYHS	DS	Local	Coarse	1.000	0	10,559	25,966	10,560
DYHS	DS	Local	Coarse	0.999	0	6,592	14,056	6,594
DYHS	DS	Local	Fine	1.000	0	13,686	32,650	13,687
DYHS	DS	Local	Fine	0.999	0	9,438	19,416	9,442
DYHS	DS	Mless	Coarse	1.000	0	14,025	34,687	14,029
DYHS	DS	Mless	Coarse	0.999	0	102,389	212,689	102,668
DYHS	DS	Mless	Fine	1.000	0	17,422	42,036	17,425
DYHS	DS	Mless	Fine	0.999	0	66,648	138,358	66,811
HZ	HZ	LBFGS	—	1.000	0	19,982	39,846	20,104
HZ	HZ	Full	Coarse	1.000	0	37,048	75,623	44,422
HZ	HZ	Full	Coarse	0.999	0	19,398	38,903	22,485
HZ	HZ	Full	Fine	1.000	0	26,898	55,242	33,572
HZ	HZ	Full	Fine	0.999	0	134,273	267,601	139,130
HZ	HZ	Local	Coarse	1.000	0	15,004	30,626	18,036
HZ	HZ	Local	Coarse	0.999	0	50,374	100,251	51,092
HZ	HZ	Local	Fine	1.000	0	12,148	24,542	14,002
HZ	HZ	Local	Fine	0.999	0	113,034	225,892	113,464
HZ	HZ	Mless	Coarse	1.000	0	13,218	26,669	15,068
HZ	HZ	Mless	Coarse	0.999	0	434,477	868,612	435,375
HZ	HZ	Mless	Fine	1.000	0	13,439	27,029	14,757
HZ	HZ	Mless	Fine	0.999	0	361,991	722,935	363,519
HZ	DS	LBFGS	—	1.000	12	130,051	267,255	130,307
HZ	DS	Full	Coarse	1.000	0	41,454	105,927	41,457
HZ	DS	Full	Coarse	0.999	0	32,023	74,727	32,047
HZ	DS	Full	Fine	1.000	0	26,062	64,157	26,064
HZ	DS	Full	Fine	0.999	0	22,723	52,630	22,752
HZ	DS	Local	Coarse	1.000	0	10,422	25,056	10,433
HZ	DS	Local	Coarse	0.999	0	10,634	22,358	10,688
HZ	DS	Local	Fine	1.000	0	13,238	30,213	13,255
HZ	DS	Local	Fine	0.999	0	7,976	16,631	8,012
HZ	DS	Mless	Coarse	1.000	0	15,578	35,038	15,602
HZ	DS	Mless	Coarse	0.999	0	40,658	84,202	41,232
HZ	DS	Mless	Fine	1.000	0	22,808	51,188	22,859
HZ	DS	Mless	Fine	0.999	0	30,613	63,786	31,002

Table A.14 — *Results for problem* `NCCS` *(level 7, 130,050 variables).*

β	linesearch	select	order	τ	status	iter	#f	#g
QN	HZ	LBFGS	—	1.000	0	29,619	59,220	88,839
QN	HZ	Full	Coarse	1.000	0	31,022	35,849	66,871
QN	HZ	Full	Coarse	0.999	0	22,076	28,678	50,754
QN	HZ	Full	Fine	1.000	0	29,235	33,036	62,271
QN	HZ	Full	Fine	0.999	0	20,398	25,958	46,356
QN	HZ	Local	Coarse	1.000	0	13,121	15,415	28,536
QN	HZ	Local	Coarse	0.999	0	61,901	122,800	184,701
QN	HZ	Local	Fine	1.000	0	19,041	22,983	42,024
QN	HZ	Local	Fine	0.999	0	73,108	145,307	218,415
QN	HZ	Mless	Coarse	1.000	0	13,547	17,301	30,848
QN	HZ	Mless	Coarse	0.999	0	439,575	877,685	329,315
QN	HZ	Mless	Fine	1.000	0	14,228	18,900	33,128
QN	HZ	Mless	Fine	0.999	0	421,398	841,370	315,692
QN	DS	LBFGS	—	1.000	12	130,051	133,157	130,051
QN	DS	Full	Coarse	1.000	0	40,156	64,776	40,157
QN	DS	Full	Coarse	0.999	0	24,111	34,121	24,112
QN	DS	Full	Fine	1.000	0	29,683	47,535	29,684
QN	DS	Full	Fine	0.999	0	24,529	35,038	24,530
QN	DS	Local	Coarse	1.000	0	14,849	21,873	14,850
QN	DS	Local	Coarse	0.999	0	5,329	6,275	5,330
QN	DS	Local	Fine	1.000	0	13,445	18,705	13,447
QN	DS	Local	Fine	0.999	0	6,113	7,252	6,114
QN	DS	Mless	Coarse	1.000	0	14,804	21,648	14,814
QN	DS	Mless	Coarse	0.999	0	56,213	61,958	56,409
QN	DS	Mless	Fine	1.000	0	16,572	23,005	16,578
QN	DS	Mless	Fine	0.999	0	25,522	28,347	25,609
DYHS	HZ	LBFGS	—	1.000	0	21,286	42,451	21,409
DYHS	HZ	Full	Coarse	1.000	0	38,748	79,016	46,265
DYHS	HZ	Full	Coarse	0.999	0	22,177	44,288	25,831
DYHS	HZ	Full	Fine	1.000	0	28,494	58,710	36,273
DYHS	HZ	Full	Fine	0.999	0	26,542	53,582	31,424
DYHS	HZ	Local	Coarse	1.000	0	14,398	29,487	17,440
DYHS	HZ	Local	Coarse	0.999	0	44,878	89,276	45,557
DYHS	HZ	Local	Fine	1.000	0	14,097	28,562	16,634
DYHS	HZ	Local	Fine	0.999	0	70,959	141,846	71,245
DYHS	HZ	Mless	Coarse	1.000	0	13,707	27,649	15,720
DYHS	HZ	Mless	Coarse	0.999	0	410,980	820,258	413,276
DYHS	HZ	Mless	Fine	1.000	0	14,738	29,659	16,387
DYHS	HZ	Mless	Fine	0.999	0	431,683	862,480	433,157
DYHS	DS	LBFGS	—	1.000	12	130,051	260,339	130,051
DYHS	DS	Full	Coarse	1.000	0	41,178	108,083	41,179
DYHS	DS	Full	Coarse	0.999	0	25,856	62,258	25,857
DYHS	DS	Full	Fine	1.000	0	32,108	83,554	32,109
DYHS	DS	Full	Fine	0.999	0	29,569	65,520	29,570
DYHS	DS	Local	Coarse	1.000	0	13,284	32,614	13,285
DYHS	DS	Local	Coarse	0.999	0	5,982	12,913	5,983
DYHS	DS	Local	Fine	1.000	0	13,994	33,510	13,995
DYHS	DS	Local	Fine	0.999	0	6,372	13,603	6,373
DYHS	DS	Mless	Coarse	1.000	0	16,287	40,373	16,290
DYHS	DS	Mless	Coarse	0.999	0	112,012	232,746	112,346
DYHS	DS	Mless	Fine	1.000	0	18,368	44,421	18,372
DYHS	DS	Mless	Fine	0.999	0	51,987	108,306	52,128
HZ	HZ	LBFGS	—	1.000	0	21,374	42,518	21,606
HZ	HZ	Full	Coarse	1.000	0	32,607	66,663	39,382
HZ	HZ	Full	Coarse	0.999	0	19,242	38,429	22,873
HZ	HZ	Full	Fine	1.000	0	31,597	65,133	39,892
HZ	HZ	Full	Fine	0.999	0	20,454	41,148	24,340
HZ	HZ	Local	Coarse	1.000	0	13,917	28,439	16,908
HZ	HZ	Local	Coarse	0.999	0	90,512	180,394	91,390
HZ	HZ	Local	Fine	1.000	0	13,131	26,560	15,145
HZ	HZ	Local	Fine	0.999	0	102,106	204,120	102,495
HZ	HZ	Mless	Coarse	1.000	0	12,469	25,240	14,287
HZ	HZ	Mless	Coarse	0.999	0	424,130	847,645	425,300
HZ	HZ	Mless	Fine	1.000	0	14,374	28,896	16,061
HZ	HZ	Mless	Fine	0.999	0	413,701	826,422	415,240
HZ	DS	LBFGS	—	1.000	12	130,051	267,428	130,337
HZ	DS	Full	Coarse	1.000	0	42,900	109,942	42,901
HZ	DS	Full	Coarse	0.999	0	27,659	64,424	27,674
HZ	DS	Full	Fine	1.000	0	32,117	79,428	32,121
HZ	DS	Full	Fine	0.999	0	26,712	61,945	26,743
HZ	DS	Local	Coarse	1.000	0	15,361	36,424	15,378
HZ	DS	Local	Coarse	0.999	0	8,225	17,247	8,279
HZ	DS	Local	Fine	1.000	0	12,422	28,276	12,429
HZ	DS	Local	Fine	0.999	0	9,867	20,735	9,901
HZ	DS	Mless	Coarse	1.000	0	14,768	34,142	14,820
HZ	DS	Mless	Coarse	0.999	0	53,127	110,173	53,910
HZ	DS	Mless	Fine	1.000	0	36,372	80,519	36,474
HZ	DS	Mless	Fine	0.999	0	31,030	64,622	31,528

Table A.15 — *Results for problem* `NCCO` *(level 7, 130,050 variables).*

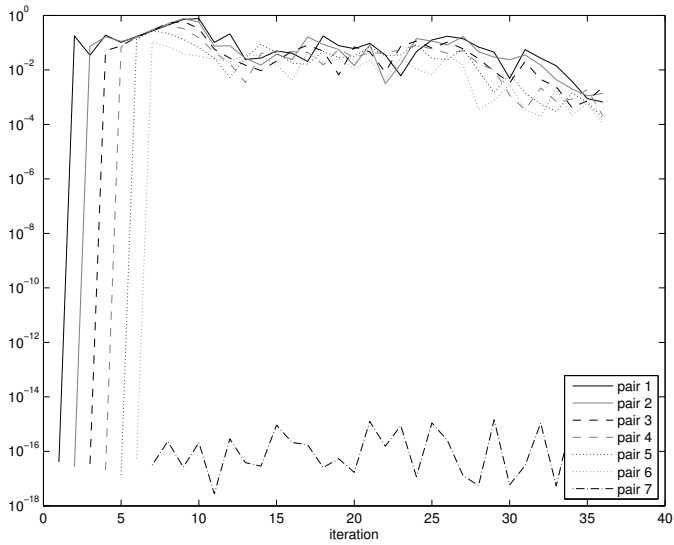
β	linesearch	select	order	τ	status	iter	#f	#g
QN	HZ	LBFGS	—	1.000	0	25,283	50,538	75,821
QN	HZ	Full	Coarse	1.000	0	28,261	34,072	62,333
QN	HZ	Full	Coarse	0.999	0	12,786	23,779	36,565
QN	HZ	Full	Fine	1.000	0	12,713	15,123	27,836
QN	HZ	Full	Fine	0.999	0	16,839	22,622	39,461
QN	HZ	Local	Coarse	1.000	0	9,692	11,962	21,654
QN	HZ	Local	Coarse	0.999	0	13,773	27,479	41,252
QN	HZ	Local	Fine	1.000	0	9,108	11,273	20,381
QN	HZ	Local	Fine	0.999	0	22,490	44,837	67,327
QN	HZ	Mless	Coarse	1.000	0	8,398	10,854	19,252
QN	HZ	Mless	Coarse	0.999	0	30,066	60,050	90,116
QN	HZ	Mless	Fine	1.000	0	7,081	9,457	16,538
QN	HZ	Mless	Fine	0.999	0	15,482	30,910	46,392
QN	DS	LBFGS	—	1.000	0	14,348	14,706	14,349
QN	DS	Full	Coarse	1.000	0	10,952	17,632	10,953
QN	DS	Full	Coarse	0.999	0	15,984	22,030	15,985
QN	DS	Full	Fine	1.000	0	16,559	25,562	16,560
QN	DS	Full	Fine	0.999	0	15,474	21,580	15,475
QN	DS	Local	Coarse	1.000	0	7,994	12,079	7,995
QN	DS	Local	Coarse	0.999	0	6,386	6,898	6,390
QN	DS	Local	Fine	1.000	0	7,279	10,228	7,280
QN	DS	Local	Fine	0.999	0	5,110	5,590	5,113
QN	DS	Mless	Coarse	1.000	0	4,316	6,350	4,319
QN	DS	Mless	Coarse	0.999	0	2,446	2,688	2,456
QN	DS	Mless	Fine	1.000	0	6,979	9,632	6,981
QN	DS	Mless	Fine	0.999	0	4,017	4,435	4,028
DYHS	HZ	LBFGS	—	1.000	0	22,442	44,885	22,443
DYHS	HZ	Full	Coarse	1.000	0	16,034	33,456	19,829
DYHS	HZ	Full	Coarse	0.999	0	16,755	33,810	17,430
DYHS	HZ	Full	Fine	1.000	0	17,505	36,005	21,415
DYHS	HZ	Full	Fine	0.999	0	16,080	32,601	18,037
DYHS	HZ	Local	Coarse	1.000	0	7,613	16,032	9,539
DYHS	HZ	Local	Coarse	0.999	0	26,900	53,828	26,966
DYHS	HZ	Local	Fine	1.000	0	10,105	20,365	11,257
DYHS	HZ	Local	Fine	0.999	0	16,662	33,328	16,678
DYHS	HZ	Mless	Coarse	1.000	0	4,759	9,886	5,700
DYHS	HZ	Mless	Coarse	0.999	0	33,220	66,453	33,258
DYHS	HZ	Mless	Fine	1.000	0	7,851	15,779	8,549
DYHS	HZ	Mless	Fine	0.999	0	21,431	42,884	21,471
DYHS	DS	LBFGS	—	1.000	0	12,367	24,756	12,368
DYHS	DS	Full	Coarse	1.000	0	17,200	45,479	17,201
DYHS	DS	Full	Coarse	0.999	0	14,249	33,704	14,250
DYHS	DS	Full	Fine	1.000	0	17,764	45,739	17,765
DYHS	DS	Full	Fine	0.999	0	19,856	47,571	19,857
DYHS	DS	Local	Coarse	1.000	0	4,608	11,636	4,609
DYHS	DS	Local	Coarse	0.999	0	3,198	6,680	3,199
DYHS	DS	Local	Fine	1.000	0	9,055	21,615	9,056
DYHS	DS	Local	Fine	0.999	0	3,300	6,968	3,302
DYHS	DS	Mless	Coarse	1.000	0	6,605	16,550	6,606
DYHS	DS	Mless	Coarse	0.999	0	7,204	14,924	7,234
DYHS	DS	Mless	Fine	1.000	0	9,260	22,333	9,264
DYHS	DS	Mless	Fine	0.999	0	7,265	15,135	7,277
HZ	HZ	LBFGS	—	1.000	0	22,442	44,885	22,443
HZ	HZ	Full	Coarse	1.000	0	9,776	20,243	11,937
HZ	HZ	Full	Coarse	0.999	0	12,906	25,936	13,266
HZ	HZ	Full	Fine	1.000	0	15,266	31,419	18,701
HZ	HZ	Full	Fine	0.999	0	12,196	24,488	12,621
HZ	HZ	Local	Coarse	1.000	0	5,098	10,736	6,411
HZ	HZ	Local	Coarse	0.999	0	9,410	18,821	9,430
HZ	HZ	Local	Fine	1.000	0	5,760	11,708	6,537
HZ	HZ	Local	Fine	0.999	0	15,309	30,622	15,332
HZ	HZ	Mless	Coarse	1.000	0	4,467	9,296	5,377
HZ	HZ	Mless	Coarse	0.999	0	18,425	36,851	18,439
HZ	HZ	Mless	Fine	1.000	0	7,271	14,666	7,965
HZ	HZ	Mless	Fine	0.999	0	22,601	45,203	22,619
HZ	DS	LBFGS	—	1.000	0	12,417	25,534	12,441
HZ	DS	Full	Coarse	1.000	0	16,401	42,098	16,402
HZ	DS	Full	Coarse	0.999	0	13,789	31,642	13,800
HZ	DS	Full	Fine	1.000	0	14,625	35,445	14,627
HZ	DS	Full	Fine	0.999	0	12,178	27,812	12,200
HZ	DS	Local	Coarse	1.000	0	4,657	11,261	4,662
HZ	DS	Local	Coarse	0.999	0	3,357	7,093	3,384
HZ	DS	Local	Fine	1.000	0	8,977	20,563	8,988
HZ	DS	Local	Fine	0.999	0	4,149	8,706	4,172
HZ	DS	Mless	Coarse	1.000	0	3,276	7,639	3,280
HZ	DS	Mless	Coarse	0.999	0	4,795	9,994	4,871
HZ	DS	Mless	Fine	1.000	0	10,837	23,856	10,853
HZ	DS	Mless	Fine	0.999	0	5,260	11,116	5,353

Table A.16 — *Results for problem* `MOREBV` *(level 8, 261,121 variables).*

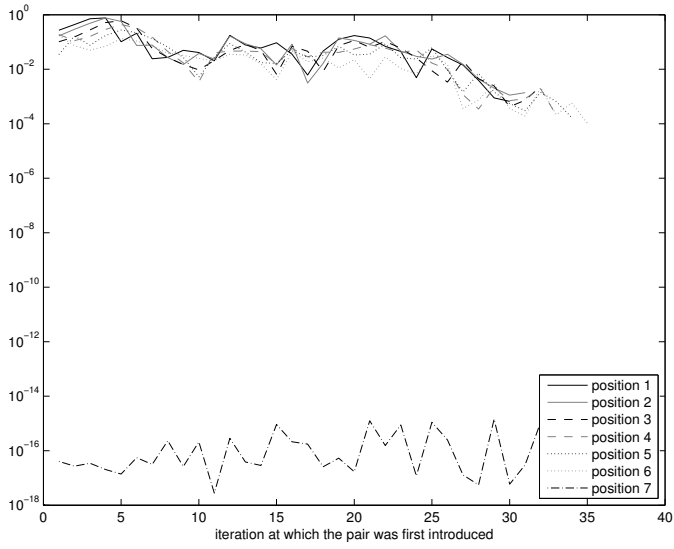
Appendix B

Additional results on the L-BFGS method behaviour

We present here complementary results of our study on the L-BFGS behaviour. We indeed consider some optimization problems from the CUTEr library (Gould *et al.*, 2003*a*), that are not of the multilevel kind, aiming at finding out whether the trends observed with multilevel problems were of more general scope. We report in the following figures the results for the BIGGS6 (which has 6 variables). They show more unclear trends than with the multilevel problems considered in Subsection 6.6.1.



(a) *Pairs used at the same iteration are vertically aligned.*



(b) *All uses of the same pair are vertically aligned.*

Figure B.1 — *Relative error* $\varepsilon(s_i, y_i)$ *for the secant pairs generated by L-BFGS on problem* `BIGGS6` *(*$m = 7$*, Dennis-Schnabel linesearch).*

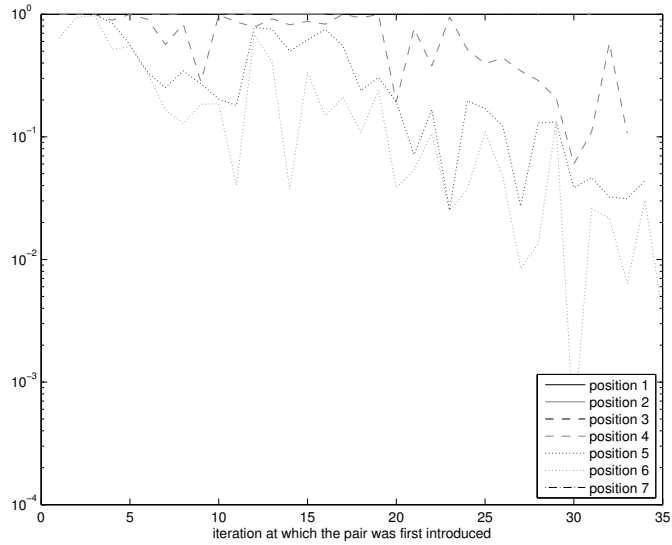
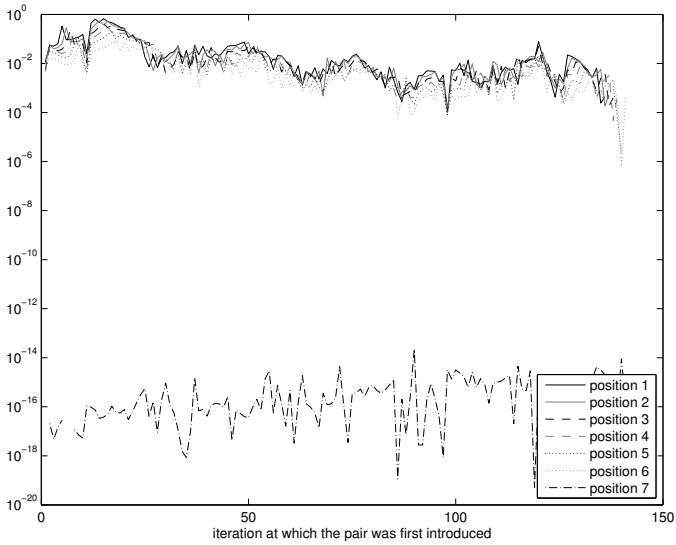
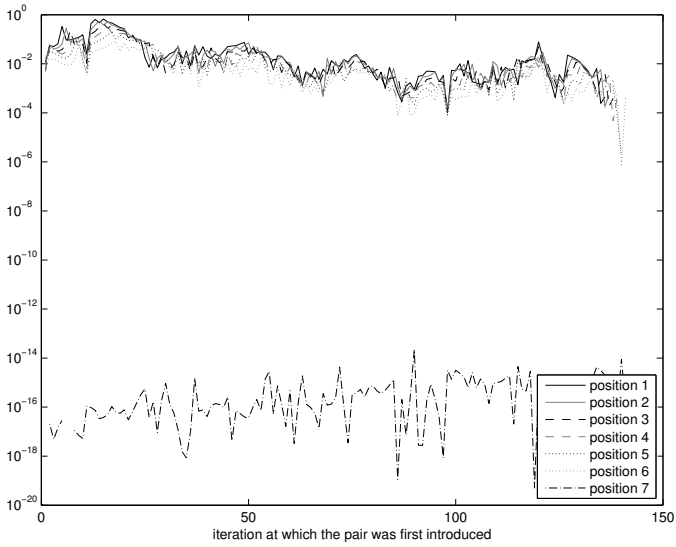


Figure B.2 — *Conjugation of direction $s_{k,i}$ with respect to the newer directions $s_{k,j}$ $(j > i)$ for the problem* `BIGGS6` *($m = 7$, Dennis-Schnabel linesearch). All uses of the same pair are vertically aligned.*



(a) *Moré-Thuente linesearch with* $\sigma = 0.9$.



(b) *Moré-Thuente linesearch with* $\sigma = 0.7$.

Figure B.3 — *Relative error* $\varepsilon(s_i, y_i)$ *for problem* `BIGGS6` *(*$m = 7$*). All uses of the same pair are vertically aligned.*

Appendix C

Computational details on the penalized Hessian updates

We first consider an alternative (but more restrictive) way to solve the penalized variational problem (6.17) on page 145 (Section C.1). From this alternative formulation, we then derive the counterpart of the PSB, DFP and BFGS updates in the context of penalized secant equations (Section C.2).

C.1 Alternative way to solve the penalized variational problem

We assume that the matrix A (from equation (6.20) on page 146) is normal, which is the case for instance when $m = 1$ or $W = I$. This matrix is thus orthogonally diagonalizable:

$$A = \begin{pmatrix} V & V_\perp \end{pmatrix} \begin{pmatrix} \Lambda & \\ & I_w \end{pmatrix} \begin{pmatrix} V^T \\ V_\perp^T \end{pmatrix}, \tag{C.1}$$

with $w = n - m$, and where the columns of the $n \times w$ matrix $V_\perp$ form an orthonormal basis of the eigenspace $\mathcal{S}^\perp$, which is associated with the eigenvalue 1, and V contains the normed eigenvectors associated with the other eigenvalues. Note that the columns of $\bar{S}$, V and $\bar{Z}$ hence span the same space $\mathcal{S}$. Using the Kronecker sum $\oplus$ (defined by $A \oplus B = A \otimes I + I \otimes B$), equation (6.20) rewrites as

$$(A \oplus A) \operatorname{vec} E = (\bar{R} \otimes \bar{Z} + \bar{Z} \otimes \bar{R}) \operatorname{vec} I_m.$$

This linear system (in the variable $\operatorname{vec} E$) is easily invertible since its spectral decomposition is known and we obtain

$$\begin{aligned}
&\operatorname{vec} E \\
&= \begin{pmatrix} V^T \otimes V^T \\ V^T \otimes V_\perp^T \\ V_\perp^T \otimes V^T \\ V_\perp^T \otimes V_\perp^T \end{pmatrix}^T \operatorname{diag} \begin{pmatrix} \Lambda \oplus \Lambda \\ \Lambda \oplus I_w \\ I_w \oplus \Lambda \\ I_w \oplus I_w \end{pmatrix}^{-1} \begin{pmatrix} V^T \otimes V^T \\ V^T \otimes V_\perp^T \\ V_\perp^T \otimes V^T \\ V_\perp^T \otimes V_\perp^T \end{pmatrix} (\bar{R} \otimes \bar{Z} + \bar{Z} \otimes \bar{R}) \operatorname{vec} I_m \\
&= \begin{pmatrix} V^T \otimes V^T \\ V^T \otimes V_\perp^T \\ V_\perp^T \otimes V^T \\ V_\perp^T \otimes V_\perp^T \end{pmatrix}^T \operatorname{diag} \begin{pmatrix} \Gamma \\ \Theta \otimes I_w \\ I_w \otimes \Theta \\ \frac{1}{2} I_w \otimes I_w \end{pmatrix} \begin{pmatrix} V^T\bar{R} \otimes V^T\bar{Z} + V^T\bar{Z} \otimes V^T\bar{R} \\ V^T\bar{Z} \otimes V_\perp^T\bar{R} \\ V_\perp^T\bar{R} \otimes V^T\bar{Z} \\ 0 \end{pmatrix} \operatorname{vec} I_m \\
&= \big[(V \otimes V)\Gamma_m (V^T\bar{R} \otimes V^T\bar{Z} + V^T\bar{Z} \otimes V^T\bar{R}) \\
&\qquad\qquad + J_1\bar{Z} \otimes J_2\bar{R} + J_2\bar{R} \otimes J_1\bar{Z}\big] \operatorname{vec} I_m
\end{aligned}$$

because $V_\perp^T Z = 0$, and where $\Gamma = (\Lambda \oplus \Lambda)^{-1}$, $\Theta = (I_m + \Lambda)^{-1}$, $J_1 = V\Theta V^T$ and $J_2 = V_\perp V_\perp^T$. As a consequence,

$$E = V(\operatorname{mat}(\operatorname{diag} \Gamma) \bullet V^T(\bar{R}\bar{Z}^T + \bar{Z}\bar{R}^T)V)V^T + J_1\bar{Z}\bar{R}^T J_2 + J_2\bar{R}\bar{Z}^T J_1$$

where $\operatorname{mat} : \mathbb{R}^{m^2} \to \mathbb{R}^{m \times m} : x \mapsto \operatorname{mat} x$, such that $\operatorname{vec}(\operatorname{mat} x) = x$. Remark now that $V^T(I_n + A)^{-1}V = (I_m + \Lambda)^{-1} = \Theta$, and recall that

$$(I_m + A)^{-1} = (2I_n + \bar{Z}\bar{S}^T)^{-1} = \tfrac{1}{2}\left(I_n - \bar{Z}(2I_m + \bar{S}^T\bar{Z})^{-1}\bar{S}^T\right)$$

by the Sherman-Morrison-Woodbury formula. Then,

$$\begin{aligned}
J_1\bar{Z} &= \tfrac{1}{2}\bar{Z}[I_m - (2I_m + \bar{S}^T\bar{Z})^{-1}(\bar{S}^T\bar{Z})], \\
J_2\bar{R} &= \bar{R} - \bar{S}(\bar{S}^T\bar{S})^{-1}(\bar{S}^T\bar{R}),
\end{aligned}$$

as VV^T is the orthogonal projection onto $\mathcal{S}$ and J_2 is the orthogonal projection onto $\mathcal{S}^\perp$.

Numerical cost. — In order to compute the eigenvalues Λ and eigenvectors V for the correction with multiple secant equations, we perform a thin QR factorization of $\bar{S} = QX$, where Q is an $n \times m$ matrix with orthonormal columns and X is an $m \times m$ upper triangular matrix. This decomposition can be updated at each iteration in $O(nm + m^2)$ flops (see Section 12.5.2 in Golub and Van Loan, 1996). Then, we perform an SVD factorization on the matrix $X = U_1 \Sigma U_2^T$, where U_1 and U_2 are $m \times m$ orthogonal matrices and Σ is an $m \times m$ diagonal matrix. This requires $O(m^3)$ additional flops. Finally, we obtain the eigenvectors $V = QU_1$ ($2nm^2$ flops) and eigenvalues $\Lambda_m = \sqrt{\Sigma}$.

C.2 Derivation of single-secant penalized updates

Consider the case of a single secant pair ($m = 1$). Then, in the previous formulation, we have $\Gamma = \frac{1}{2}\Lambda^{-1}$ and $J_1\bar{Z} = \Theta\bar{Z}$, and thus

$$E = \tfrac{1}{2}\Lambda^{-1}(VV^T\bar{R}\bar{Z}^T + \bar{Z}\bar{R}^TVV^T + (1+\Lambda)^{-1}(J_2\bar{R}\bar{Z}^T + \bar{Z}\bar{R}^TJ_2)$$

because $VV^T\bar{Z} = \bar{Z}$. Moreover, $V = s_1/\|s_1\|$ and $\Lambda = 1 + \omega_1\|s_1\|^2$, and dropping the subscripts, we thus obtain

$$\begin{aligned}\operatorname{vec} E = \Bigg(&\frac{1}{2\omega^{-1} + 2\|s\|^2}(\mathrm{P}_s\, r \otimes z + z \otimes \mathrm{P}_s\, r) \\ &+ \frac{1}{2\omega^{-1} + \|s\|^2}(z \otimes (I_n - \mathrm{P}_s)r + (I_n - \mathrm{P}_s)r \otimes z)\Bigg) \operatorname{vec} I_m.\end{aligned}$$

where $\mathrm{P}_s \overset{\text{def}}{=} VV^T = \frac{ss^T}{\|s\|^2}$ is the orthogonal projection onto $\operatorname{span}(s)$.

Penalized PSB update. — If we choose $W = I$, then $z = s$, yielding the penalized PSB update:

$$\begin{aligned}B^+ &= B + \frac{1}{2\omega^{-1} + 2\|s\|^2}(sr^T\,\mathrm{P}_s + \mathrm{P}_s\, rs^T) \\ &\qquad + \frac{1}{2\omega^{-1} + \|s\|^2}((I_n - \mathrm{P}_s)rs^T + sr^T(I_n - \mathrm{P}_s)) \\ &= B + \frac{rs^T + sr^T}{2\omega^{-1} + \|s\|^2} + \left(\frac{1}{\omega^{-1} + \|s\|^2} - \frac{2}{2\omega^{-1} + \|s\|^2}\right)\frac{s^Tr}{\|s\|^2}ss^T.\end{aligned}$$

Penalized DFP update. — If we now choose the matrix W such that $\hat{W}s = y$, then $z = y$, yielding the penalized DFP update:

$$\begin{aligned}B^+ &= B + \frac{1}{2\omega^{-1} + 2\|s\|^2}(yr^T\,\mathrm{P}_s + \mathrm{P}_s\, ry^T) \\ &\qquad + \frac{1}{2\omega^{-1} + \|s\|^2}((I_n - \mathrm{P}_s)ry^T + yr^T(I_n - \mathrm{P}_s)) \\ &= B + \frac{ry^T + yr^T}{2\omega^{-1} + \|s\|^2} + \left(\frac{1}{\omega^{-1} + \|s\|^2} - \frac{2}{2\omega^{-1} + \|s\|^2}\right)\frac{s^Tr}{\|s\|^2}yy^T.\end{aligned}$$

Penalized BFGS update. — Finally, the penalized inverse BFGS formula is obtained by duality from its direct DFP counterpart as indicated in Subsection 6.6.3.

Appendix D

Specification of the RMTR package

D.1 Summary

ATTRIBUTES — **Versions:** `GALAHAD_RMTR_single`, `GALAHAD_RMTR_double`. **Uses:** `GALAHAD_LTS`, `GALAHAD_SORT`, `GALAHAD_SPECFILE`, `GALAHAD_SYMBOLS`, `*NRM2`, `*DOT`. **Date:** June 2010. **Origin:** V. Malmedy[1,2], Ph. L. Toint[1], and D. Tomanos[1,3]. [1]: University of Namur (FUNDP); [2]: Fonds de la Recherche Scientifique (FNRS); [3]: Fonds de la Recherche pour l'Industrie et l'Agriculture (FRIA). **Language:** Fortran 95 + TR 15581 or Fortran 2003.

D.2 How to use the package

D.2.1 Matrix storage formats

Because multilevel problems are typically large, the matrices must be stored in sparse format, of which two types are allowed. In the sparse co-ordinate format, only the nonzero entries of the matrices are stored. For example, for the l-th entry of the matrix A, its row index i, column index j and value A_{ij} are stored in the l-th components of the integer arrays `A_row`, `A_col` and real array `A_val`. The order is unimportant, but the total number of entries `A_size` is also required. The user also has the possibility to store the Hessian in a compressed sparse row format. In this format, the matrix A is still stored in the integer arrays `A_row`, `A_col` and real array `A_val`. But the i-th entry of array `A_row` (which length is the number of rows of the matrix plus one) now corresponds to the index of the first component in the other two vectors where an element of the i-th line is stored. The array `A_col` then contains the

column index and `A_val` contains the value of the entry. Note that the order of the rows is important for this format.

D.2.2 The GALAHAD symbols

As several of the GALAHAD packages, RMTR makes use of "symbols" that are publicly available in the `GALAHAD_SYMBOLS` module. These symbols are conventional names given to specific integer values, and allow a more natural specification of the various options and parameters of the package. Each symbol provided in the `GALAHAD_SYMBOLS` module is of the form `GALAHAD_NAME`, where `NAME` is the name of the symbol. For clarify and conciseness, a symbol will be represented by `GALAHAD_NAME` (in sans-serif upper case font) in what follows. See Section D.4 to see how symbols may be used in the program unit that calls the RMTR subroutines.

D.2.3 The derived data types

In addition to the multilevel problem data type, two derived data types are accessible from the package.

D.2.3.1 The multilevel problem type

The derived data type `RMTR_problem_type` is used to hold all the information about the multilevel structure of the problem. It is a double linked list where each component of the structure contains all the information about a particular level i and is organized as follows.

`level` is a scalar variable of type default `INTEGER`, that holds the current level index i.

`nbvar` is a scalar variable of type default `INTEGER`, that holds the number of variables at the current level i, that is the number of variables used to represent each field of the considered problem.

`x` is a rank-one allocatable array of dimension `nbvar` and type `REAL` (double precision in `GALAHAD_RMTR_double`), that hold the values of the problem variables at the current iterate of level i. The j-th component of `x`, $j = 1, \ldots,$ `nbvar`, contains x_j .

`x_start` is a rank-one allocatable array of dimension `nbvar` and type `REAL` (double precision in `GALAHAD_RMTR_double`), that holds the values of the problem variables at the starting point of the current minimization sequence at the considered level i.

`g` is a rank-one allocatable array of dimension `nbvar` and type `REAL` (double precision in `GALAHAD_RMTR_double`), that holds the values of the objective gradient at the current iterate of level i.

`H_ne` is a scalar variable of type default `INTEGER`, that holds the number of nonzero elements in the objective Hessian of level i.

`H_val, H_row, H_col` are rank-one allocatable arrays of type `REAL` (double precision in `GALAHAD_RMTR_double`), default `INTEGER` and default `INTEGER`, respectively, that hold the values of the objective Hessian approximation at the current iterate of level i in CSR storage.

`R_ne` is a scalar variable of type default `INTEGER`, that holds the number of nonzero elements in the restriction operator to the coarser level (that is the operator from the current level i to the coarser level $i-1$).

`R_val, R_row, R_col` are rank-one allocatable arrays of type `REAL` (double precision in `GALAHAD_RMTR_double`), default `INTEGER` and default `INTEGER`, respectively, that hold the values of the restriction operator to the coarser level (that is the operator from the current level i to the coarser level $i-1$) in CSR storage.

`P_ne` is a scalar variable of type default `INTEGER`, that holds the number of nonzero elements in the prolongation operator to the finer level (that is the operator from the current level i to the finer level $i+1$).

`P_val, P_row, P_col` are rank-one allocatable arrays of type `REAL` (double precision in `GALAHAD_RMTR_double`), default `INTEGER` and default `INTEGER`, respectively, that hold the values of the prolongation operator to the finer level (that is the operator from the current level i to the finer level $i+1$) in CSR storage.

`sigma` is a scalar variable of type default `REAL`, that holds the 1-norm of the restriction operator stored in the current structure.

`lower_bound` is a rank-one allocatable array of dimension `nbvar` and type `REAL` (double precision in `GALAHAD_RMTR_double`), that holds the values of the lower-bound constraints on the problem variables at the current level i.

`upper_bound` is a rank-one allocatable array of dimension `nbvar` and type `REAL` (double precision in `GALAHAD_RMTR_double`), that holds the values of the upper-bound constraints on the problem variables at the current level i.

`nbr_iter` is a scalar variable of type default `INTEGER`, that holds the number of iterations in the current minimization sequence at level i.

`tr_radius` is a scalar variable of type `REAL` (double precision in `GALAHAD_RMTR_double`), that holds the current level trust-region radius of level i.

`eps_f` is a scalar variable of type `REAL` (double precision in `GALAHAD_RMTR_double`), that holds the required accuracy on the objective function value (for least-square problems) at level i.

`eps_g` is a scalar variable of type `REAL` (double precision in `GALAHAD_RMTR_double`), that holds the required accuracy on the criticality measure at level i.

`level_up` is a pointer to the `RMTR_problem_type` structure of the finer level (that is the structure of level $i + 1$).

`level_down` is a pointer to the `RMTR_problem_type` structure of the coarser level (that is the structure of level $i - 1$).

All these data are allocated and initialized by the subroutine `RMTR_initialize`. Note that no memory is allocated when it is not needed. Note also that other quantities that are used inside the algorithm are also stored in this structure.

D.2.3.2 The derived data type for holding control parameters

The derived data type `RMTR_control_type` is used to hold controlling data. Its components thus hold both method settings and problem features. The different components holding method settings are:

`error_printout_device` is a scalar variable of type default `INTEGER`, that holds the unit number associated with the device used for error output. The default is `error_printout_device = 6`.

`printout_device` is a scalar variable of type default `INTEGER`, that holds the unit number associated with the device used for output. The default is `printout_device = 6`.

`print_level` is a symbolic variable, that holds the level of printout requested by the user. The variable may take the values `GALAHAD_SILENT`, `GALAHAD_SUMMARY`, `GALAHAD_TRACE`, `GALAHAD_ACTION`, `GALAHAD_DETAILS`, `GALAHAD_DEBUG` and `GALAHAD_CRAZY`. These values are described in detail in Subsection D.2.8. The default is `print_level = GALAHAD_TRACE`.

`start_printing_at_iteration` is a scalar variable of type default `INTEGER`, that holds the index of the first RMTR iteration at which printing must occur. The default is `start_printing_at_iteration = 0` (print from initialization on).

`stop_printing_at_iteration` is a scalar variable of type default `INTEGER`, that holds the index of the last RMTR iteration at which printing must occur. If negative, printing does not stop once started. The default is `stop_printing_at_iteration = -1` (always print once started).

`display_equivalent_evaluations` is a scalar variable of type default `LOGICAL` that has the value `.TRUE.` iff the program needs to print a summary of the

total equivalent amount of finest level work at the end of the algorithm. Note that this summary is printed only if the `print_level` parameter is larger than `GALAHAD_SUMMARY`. The default is `display_equivalent_evaluations = .TRUE.`.

`display_options` is a scalar variable of type default `LOGICAL` that has the value `.TRUE.` iff the program needs to print the options (used by the method) in the `RMTR_initialize` subroutine. Note that the options are printed only if the `print_level` parameter is larger than `GALAHAD_SUMMARY`. The default is `display_options = .TRUE.`.

`save_solution` is a scalar variable of type default `LOGICAL` that has the value `.TRUE.` iff the program needs to store the solution of the problem in a file (which may be specified by the user in the problem specification file with the parameter `solution_file`; see Subsection D.2.5.2). The solution is saved at the beginning of the `RMTR_terminate` subroutine. The default is `save_solution = .TRUE.`.

`criticality_threshold` is a scalar variable of type `REAL` (double precision in `GALAHAD_RMTR_double`), that specifies an accuracy threshold such that the RMTR algorithm is successfully terminated if the criticality measure on the finest level is under that threshold. The default is `criticality_threshold` = 10^{-6}.

`function_threshold` is a scalar variable of type `REAL` (double precision in `GALAHAD_RMTR_double`), that specifies an accuracy threshold such that the RMTR algorithm is successfully terminated if the objective function value on the finest level is under that threshold (useful for least-square problems). The default is `function_threshold` = 10^{20} (disabling normally the stopping criterion on the objective function value).

`truncated_conjugate_gradient_accuracy` is a scalar variable of type `REAL` (double precision in `GALAHAD_RMTR_double`), that specifies an accuracy threshold such that the PTCG minimization of the Taylor model (see Gratton *et al.*, 2010*b*) is successfully terminated if the model gradient is under that threshold. The default is `truncated_conjugate_gradient_accuracy` = 10^{-1}.

`maximum_number_of_iterations` is a scalar variable of type default `INTEGER`, that holds the maximum number of RMTR iterations allowed during a call to `RMTR_solve`. The default is `maximum_number_of_iterations = 1000`.

`maximum_number_of_tcg_iterations` is a scalar variable of type default `INTEGER`, that holds the maximum number of truncated-conjugate-gradient iterations during a minimization of the Taylor model (see Gratton *et al.*, 2010*b*). The default is `maximum_number_of_tcg_iterations = 5`.

`maximum_solving_time` is a scalar variable of type `REAL` (double precision in `GALAHAD_RMTR_double`), that specifies the maximum amount of time (in seconds) allowed during a call to `RMTR_solve`. If negative, no upper limit is imposed. The default is `maximum_solving_time = 3600`.

`minimum_rho_for_successful_iteration` is a scalar variable of type `REAL` (double precision in `GALAHAD_RMTR_double`), that holds the minimum ratio of achieved vs. predicted reduction for declaring a RMTR iteration successful. The default is `minimum_rho_for_successful_iteration = 0.01`.

`minimum_rho_for_very_successful_iteration` is a scalar variable of type `REAL` (double precision in `GALAHAD_RMTR_double`), that holds the minimum ratio of achieved vs. predicted reduction for declaring a RMTR iteration very successful. The default is `minimum_rho_for_very_successful_iteration = 0.9`.

`radius_reduction_factor` is a scalar variable of type `REAL` (double precision in `GALAHAD_RMTR_double`), that holds the radius reduction factor in the case of an unsuccessful iteration. The default is `radius_reduction_factor = 0.25`.

`radius_increase_factor` is a scalar variable of type `REAL` (double precision in `GALAHAD_RMTR_double`), that holds the radius increase factor in the case of a successful iteration. The default is `radius_increase_factor = 2.0`.

`maximum_radius_increase_factor` is a scalar variable of type `REAL` (double precision in `GALAHAD_RMTR_double`), that holds the radius increase factor in the case of a very successful iteration. The default is `maximum_radius_increase_factor = 3.0`.

`maximum_radius` is a scalar variable of type `REAL` (double precision in `GALAHAD_RMTR_double`), that holds the maximum trust-region radius. If negative, no upper limit is imposed. The default is `maximum_radius = -1.0`.

`initial_radius` is a scalar variable of type `REAL` (double precision in `GALAHAD_RMTR_double`), that holds the initial trust-region radius. The default is `initial_radius = 1.0`.

`forced_Hessian_evaluation_frequency` is a scalar variable of type default `INTEGER`, that holds the maximum number of iterations allowed without recomputing the Hessian. Indeed, since computing the objective Hessian is commonly one of the heaviest task of optimization algorithms, RMTR features a strategy that avoids recomputing the Hessian at each iteration (see Gratton *et al.*, 2010*b*). If zero, no forced evaluation is made except by

the automatic criterion. The default is `forced_Hessian_evaluation_frequency = 0`.

`forced_Hessian_evaluation_factor` is a scalar variable of type `REAL` (double precision in `GALAHAD_RMTR_double`), that holds the minimum ratio of achieved vs. predicted reduction under which Hessian evaluation is forced. The default is `forced_Hessian_evaluation_factor = 0.5`.

`euclidean_gradient_accuracy_for_Hessian_evaluation` is a scalar variable of type `REAL` (double precision in `GALAHAD_RMTR_double`), that holds the minimum relative accuracy of the predicted gradient (in Euclidean norm) under which Hessian reevaluation is forced. The default is `euclidean_gradient_accuracy_for_Hessian_evaluation = 0.15`.

`infinite_gradient_accuracy_for_Hessian_evaluation` is a scalar variable of type `REAL` (double precision in `GALAHAD_RMTR_double`), that holds the minimum relative accuracy of the predicted gradient (in infinity norm) under which Hessian reevaluation is forced. The default is `infinite_gradient_accuracy_for_Hessian_evaluation = 10000`.

`initialization_technique` is a symbolic variable, that holds the multilevel strategy used to solve the problem (see Gratton *et al.*, 2010*b*). The parameter may take the values:

`GALAHAD_AF` (All on Finest algorithm),

`GALAHAD_MR` (Mesh Refinement algorithm),

`GALAHAD_FM` (Full Multilevel algorithm),

`GALAHAD_MF` (Multilevel on Finest algorithm), and

`GALAHAD_FMF` (Full Multilevel on Finest algorithm).

If algorithm `GALAHAD_AF`, `GALAHAD_MF` or `GALAHAD_FMF` is chosen, the user only needs to provide information (objective function, gradient,...) about the finest level. Otherwise, the user has to provide information for all levels. The default is `initialization_technique = GALAHAD_FM`.

`cycling_style` is a symbolic variable that holds the type of recursion done by the algorithm (see Subsection 3.2.4). The parameter may take the values `GALAHAD_Wcycles` (W-cycles), `GALAHAD_Vcycles` (V-cycles) or, `GALAHAD_freecycles` (recursion is finished only when accuracy is reached). The default value is `cycling_style = GALAHAD_Vcycles`.

`coarse_model_choice_parameter` is a scalar variable of type `REAL` (double precision in `GALAHAD_RMTR_double`), that holds the minimum ratio of coarser linear decrease vs. current level linear decrease over which minimization of the coarser local model is preferred (see Subsection 3.2.1). The default value is `coarse_model_choice_parameter = 0.25`.

`linesearch` is a scalar variable of type default `INTEGER`, that holds the maximum number of additional function evaluations allowed for a linesearch procedure. The default is `linesearch = 2`.

`model_backtracking` is a scalar variable of type default `LOGICAL`, that is `.TRUE.` iff backtracking of the model along the current step is allowed after an unsuccessful iteration. The default is `model_backtracking = .TRUE.`.

`quadratic_model` is a symbolic variable, that holds the type of coarser local model used by the algorithm (see Gratton *et al.*, 2010*b*). The different possible values are

`GALAHAD_FIRST_ORDER` for a first-order coherent coarse local model,

`GALAHAD_SECOND_ORDER` for a second-order coherent coarse local model,

`GALAHAD_GALERKIN` which is a restricted version of the current level quadratic Taylor's model.

The default is `quadratic_model = GALAHAD_GALERKIN`.

`number_of_smoothing_cycles` is a scalar variable of type default `INTEGER`, that holds the number of smoothing cycles allowed at each minimization of the Taylor's model by a Sequential Coordinate Minimization method (see Gratton *et al.*, 2010*b*). The default is `number_of_smoothing_cycles = 7`.

`smooth_frequency` is a scalar variable of type default `INTEGER`, that holds the frequency of smoothing along the recursions. The possible values are

`GALAHAD_NEVER_SMOOTH` if no smoothing is done before or after a recursion,

`GALAHAD_SMOOTH_UP` if SCM smoothing is imposed after a recursion,

`GALAHAD_SMOOTH_DOWN` if SCM smoothing is imposed before a recursion,

`GALAHAD_ALWAYS_SMOOTH` if SCM smoothing is imposed before and after a recursion.

The default is `smooth_frequency = GALAHAD_ALWAYS_SMOOTH`.

`checkpointing_frequency` is a scalar variable of type default `INTEGER`, that holds the frequency (expressed in number of finest iterations) at which the current values of the problem's variables and the trust-region radius are saved on a checkpointing file for a possible package restart. It must be nonnegative. The default is `checkpointing_frequency = 0` (no checkpointing).

`checkpointing_file` is a scalar variable of type default `CHARACTER` of length 30, that holds the name of the file use for storing checkpointing information on disk. The default is `checkpointing_file = RMTR.sav`.

`checkpointing_device` is a scalar variable of type default `INTEGER`, that holds the number of the device that must be used for input/output of checkpointing operations. The default is `checkpointing_device = 55`.

`restart_from_checkpoint` is a scalar variable of type default `LOGICAL`, whose value is `.TRUE.` iff the initial point must be read from the checkpointing file, overriding the input value of the starting point. The default is `restart_from_checkpoint = .FALSE.`.

We refer to Subsection D.2.5.2 for a description of the components holding problem features. Default values may be obtained by calling `RMTR_initialize` (whose header is given in Subsection D.2.4), but individual components may also be changed in this routine by reading a specification file (see Subsection D.2.7). All these parameters may also be changed manually after the call to the `RMTR_initialize` subroutine.

D.2.3.3 The derived data type for holding informational parameters

The derived data type `RMTR_inform_type` is used to hold parameters that give information about the progress and needs of the algorithm. The components of `RMTR_inform_type` are:

`status` is a scalar variable of type default `INTEGER`, that gives the exit status of the algorithm. See Subsection D.2.6 for details.

`message` is a character array of 3 lines of 80 characters each, containing a description of the exit condition on exit, typically including more information than contained in status. It is printed out on device `error_print-out_device` at the end of execution unless print level is `GALAHAD_SILENT`.

`nbr_taylor` is a rank one array of type default `INTEGER`, whose length is the number of levels, and whose i-th entry holds the number of Taylor minimizations at level i, that is the number of minimizations of the Taylor's model by a PTCG procedure at that level.

`nbr_taylor_iterations` is a rank one array of type default `INTEGER`, whose length is the number of levels, and whose i-th entry holds the total number of Taylor iterations at level i, that is the total numbers of PTCG iterations at that level.

`nbr_smoothing` is a rank one array of type default `INTEGER`, whose length is the number of levels, and whose i-th entry holds the number of SCM minimizations at level i, that is the number of minimizations of the Taylor model by the SCM procedure at that level.

`nbr_cycles` is a rank one array of type default `INTEGER`, whose length is the number of levels, and whose i-th entry holds the total number of SCM cycles at level i, that is the total number of SCM iterations at that level.

`nbr_backtrackings` is a rank one array of type default `INTEGER`, whose length is the number of levels, and whose i-th entry holds the number of backtracking iterations at level i.

`nbr_f_evaluations` is a rank one array of type default `INTEGER`, whose length is the number of levels, and whose i-th entry holds the number of function evaluations at level i.

`nbr_g_evaluations` is a rank one array of type default `INTEGER`, whose length is the number of levels, and whose i-th entry holds the number of gradient evaluations at level i.

`nbr_H_evaluations` is a rank one array of type default `INTEGER`, whose length is the number of levels, and whose i-th entry holds the number of Hessian evaluations at level i.

`nbr_H_updates` is a rank one array of type default `INTEGER`, whose length is the number of levels, and whose i-th entry holds the number of Hessian evaluations at level i through the LTS package.

`nbr_H_reductions` is a rank one array of type default `INTEGER`, whose length is the number of levels, and whose i-th entry holds the number of Hessian reductions at level i, that is the number of Hessian reductions from level i to level $i-1$.

`nbr_prolongations` is a rank one array of type default `INTEGER`, whose length is the number of levels, and whose i-th entry holds the number of prolongations from level i to level $i+1$.

`nbr_restrictions` is a rank one array of type default `INTEGER`, whose length is the number of levels, and whose i-th entry holds the number of restrictions from level i to level $i-1$.

`nbr_projections` is a rank one array of type default `INTEGER`, whose length is the number of levels, and whose i-th entry holds the number of projections on the constraints at level i.

D.2.4 Argument lists and calling sequences

Access to the package requires a `USE` statement such as
Single-precision version:

```
USE GALAHAD_RMTR_SIMPLE
```

Single-precision version:

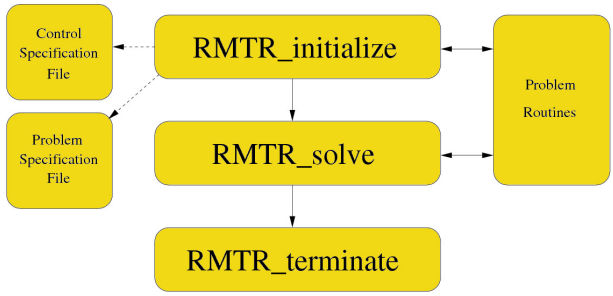


Figure D.1 — *Calling sequence of the RMTR package.*

```
USE GALAHAD_RMTR_DOUBLE
```

The package uses two specification files. The first one is a control file whose purpose is to modify the default values of the algorithmic parameters of the method. The second file defines important characteristics of the problem, such as its dimension, ... (the complete list of the problem parameters is described in Section D.2.5.2). The description and syntax of these specification files is available in autoref D.2.7.

The actual call to the package consists of three successive subroutine calls, as illustrated in Figure D.1.

1. The subroutine `RMTR_initialize` is first called by the statement

```
CALL RMTR_initialize(control, inform, problem,      &
       [controlspec], [problemspec],                &
       [STRUCT | SPARSITY],                         &
       [LOWER_BOUND] [UPPER_BOUND],                 &
       [PROLONGATION, RESTRICTION, SIZES |          &
        border_type,  coarsest_mesh_size ])
```

   where only the three first arguments are mandatory: `control` is a scalar argument of type `RMTR_control_type`, `inform` is a scalar argument of type `RMTR_inform_type`, `problem` is a pointer of type `RMTR_problem_type`. The two optional strings arguments `controlspec` and `problemspec` may contain the name of the specification files for the method settings and problem features, respectively (see Subsection D.2.7). Either the `STRUCT` or the `SPARSITY` subroutine arguments may be used to define the sparsity pattern of the Hessian. The `LOWER_BOUND` and `UPPER_BOUND` subroutine arguments may provide the lower and upper bounds, respectively, of the minimization problem. The `SIZES` subroutine argument must provide an array with the number of variables at each level of the problem, whenever the problem is not set on one of the predefined mesh structure, in which case the associated `PROLONGATION` and `RESTRICTION`

subroutine arguments must also provide the prolongation and restriction operator, respectively. A predefined mesh may be selected by setting `control%operators_type` to a positive value and providing two rank-one array of default type `INTEGER`, whose length is the problem's dimension, and that determine the type of boundary and the number of nodes at the coarsest level, respectively, in each 1-D direction of the mesh. Further information on these optional arguments is given in Subsection D.2.5.

On exit, the parameters held by `control` structure have been first initialized to their default values, and then possibly modified according to the user-provided specification files. All the components of the multilevel structure are allocated according to the problem features. A successful call to the routine `RMTR_initialize` is indicated when the component `status` of the `inform` argument has the value 0.

2. The subroutine `RMTR_solve` is called by

```
CALL RMTR_solve(control, inform, problem, FUN, GRAD, &
       [HESS])
```

The arguments `control`, `inform` and `problem` are of the same types as in the call to `RMTR_initialize`. They should have been initialized first by a call to the `RMTR_initialize` routine. The other arguments are routines to describe the problem. `FUN` is the user subroutine to compute the objective function, `GRAD` is the user subroutine to compute the objective gradient, and `HESS` is the user subroutine to compute the objective Hessian and is optional. The headers of these routines are described in Subsection D.2.5. This routine is called to solve the problem by applying the RMTR algorithm.

On exit, the solution is in the `problem%x` component. A successful call to the routine `RMTR_solve` is indicated when the component `status` of the `inform` argument has the value 0. For other return values of the `status` component, see Subsection D.2.6. The other `inform` components contain the iterations history of the algorithm as explained in Subsection D.2.3.3.

The starting point of the algorithm is also determined at the start of `RMTR_solve`, either from the user-specified values which are read in a file (the file name is specified in the problem specification file by the parameter `starting_point_file` and each line of this file has to contain the corresponding component of the starting point), or, if no such file exists, by the following simple component-wise procedure.

(a) If both lower and upper bounds are given, the starting point is computed as the mid-point between these bounds.

(b) If only a lower or upper bound is given, the starting point is computed at a distance of 1 (in infinity norm) of the specified constraint.

(c) If no bound is given, the starting point is set to 1.

(d) A small feasible random perturbation is then applied. The perturbation is uniform and of amplitude 10^{-2}.

Note that the starting point is defined at the finest level, irrespective of the actual multilevel technique used to solve the problem (if a mesh refinement or a full multilevel technique is chosen by the user, the starting point is first restricted to the coarsest level).

3. The subroutine `RMTR_terminate` is called by

```
CALL RMTR_terminate(control, inform, problem)
```

where the arguments `control`, `inform` and `problem` are of the same types as in the call to `RMTR_initialize`, and should have been initialized first by this subroutine. The `RMTR_terminate` subroutine automatically deallocates the RMTR-specific arrays. The solution is also written in a file if required by the user by the control parameter `save_solution`. In this case, each line of the file contains the corresponding component of the solution. A summary print of the iterations is also printed unless the control parameter `print_level` is set to `GALAHAD_SILENT`.

D.2.5 Information needed by the algorithm

Many multilevel problems are discretizations of an underlying infinite-dimensional problem on a grid. RMTR thus features the necessary routines for this class of problems. The sections below describe these routines and how a user has to create his routines.

D.2.5.1 Important note

Note that if the user has chosen to use the Mesh Refinement algorithm or the Full Multilevel algorithm, the problem has to be described for different levels. This means that the user has to write the routines `FUN`, `GRAD`, `HESS`, `STRUCT`, `SPARSITY`, `LOWER_BOUND` and `UPPER_BOUND` in such a way that they may be evaluated from different levels. The knowledge of this hierarchy information is important to derive the mesh refinement strategy used to obtain better starting points that leads to better numerical results. However, if this knowledge is impossible to obtain the user has to use one of the other three algorithms provided in the RMTR package. Note that the level for which the routine is computed is not an input of the routines since it is possible to derive it with from the size of the input vectors.

D.2.5.2 The problem specification

In the same spirit of the methods settings (holded by the control structure presented above), the user may provide information about his problem. These problem features are holded by the following components of the same control structure:

`level_min` is a scalar variable of type default `INTEGER`, that holds the index of the coarsest level. The default is `level_min = 1`.

`level_max` is a scalar variable of type default `INTEGER`, that holds the index of the finest level. The default is `level_max = 4`.

`lower_bound` is a scalar variable of type default `LOGICAL`, that has the value `.TRUE.` iff the problem's variables are subject to a lower bound. The default is `lower_bound = .FALSE.`.

`upper_bound` is a scalar variable of type default `LOGICAL`, that has the value `.TRUE.` iff the problem's variables are subject to an upper bound. The default is `upper_bound = .FALSE.`.

`quadratic_problem` is a scalar variable of type default `LOGICAL`, that has the value `.TRUE.` iff the problem is a quadratic problem and thus, has a constant Hessian that has not to be recomputed. The default is `quadratic_problem = .FALSE.`.

`matrix_storage` is a symbolic variable, that holds the storage format used by the used to provide the objective Hessian and/or transfer operators. Note that these matrices are internally stored in the CSR format, a conversion being performed if needed. The possible values are:

`GALAHAD_COORDINATE` for a COO storage, and

`GALAHAD_SPARSE_BY_ROWS` for a CSR storage.

The default is `matrix_storage = GALAHAD_COORDINATE`.

`half_Hessian` is a scalar variable of type default `LOGICAL`, that has the value `.TRUE.` iff only the half Hessian is computed and is stored. Note that the algorithm is in this case less efficient but requires less memory. The default is `half_Hessian = .FALSE.`.

`approximate_Hessian` is a symbolic variable, that holds the type of Hessian used by the problem. The different possible values are

`GALAHAD_EXACT_HESSIAN` if the exact Hessian is used,

`GALAHAD_LTS_STRUCT` if the Hessian is approximated by LTS using a user-supplied partial separability structure,

`GALAHAD_LTS_SPARSITY` if the Hessian is approximated by LTS using a user-supplied sparsity pattern (that is the vectors `H_row` and `H_col` stored in CSR format), and

`GALAHAD_LTS_PREDEFINED_PATTERN` if the Hessian is approximated by LTS using the predefined type of sparsity pattern selected by the option `predefined_sparsity_pattern`.

We refer to Subsection D.2.5.5 for more details on these choices. The default value is `approximate_Hessian = GALAHAD_EXACT_HESSIAN`.

`predefined_sparsity_pattern` is a scalar variable of type default `INTEGER`, that indicates the predefined sparsity pattern to approximate the objective Hessian with the LTS package. We refer to the documentation of the LTS package for a description of the possible values. This parameter is only needed if `approximate_Hessian = GALAHAD_LTS_PREDEFINED_PATTERN`.

`operators_type` is a symbolic variable, that holds the type of transfer operators used. The different possible values are:

`GALAHAD_USER` if the transfer operators are provided by the user,

`GALAHAD_LINEAR` if linear interpolation operators are always used,

`GALAHAD_LINEAR_CUBIC` if linear interpolation operators are used except for prolongating the solution at a level to define the starting point at the upper level in the mesh-refinement process, in which case a cubic interpolation operator is used instead, and

`GALAHAD_CUBIC` if cubic interpolation operators are always used.

Note that the last three choices are only valid for geometric problems. We refer to Subsection D.2.5.7 for more details on these choices. The default is `operators_type = GALAHAD_LINEAR_CUBIC`.

`starting_point_file` is a scalar variable of type default `CHARACTER` of length 30, that holds the name of the file used to store the starting point. The default is `starting_point_file = RMTR_startingpoint.dat`.

`solution_file` is a scalar variable of type default `CHARACTER` of length 30, that holds the name of the file used to store the solution. The default is `solution_file = RMTR_solution.dat`.

Initializing and changing the values of these parameters occur in the same way as for the method settings, except that it uses its own problem specification file.

D.2.5.3 The objective function

Obviously the algorithm needs to be able to evaluate the objective function at any iterate. A subroutine `FUN` has to be supplied to the algorithm as described above. Its header has to be the following:

```
SUBROUTINE FUN(x, f)
```

where

`x` is an `INTENT(IN)` array of type `REAL` (double precision in `GALAHAD_RMTR_double`) that holds the point where the objective function is computed, and

`f` is an `INTENT(OUT)` scalar of type `REAL` (double precision in `GALAHAD_RMTR_double`) that holds the computed objective value.

D.2.5.4 The objective gradient

The algorithm also needs to be able to evaluate the objective gradient at any iterate. This task has to be perform by a user-supplied subroutine `GRAD`, whose header has to be the following:

```
SUBROUTINE GRAD(x, g)
```

where

`x` is an `INTENT(IN)` array of type `REAL` (double precision in `GALAHAD_RMTR_double`) that holds the point where the objective gradient is computed, and

`g` is an `INTENT(INOUT)` array of type `REAL` (double precision in `GALAHAD_RMTR_double`) that holds the objective gradient computed.

D.2.5.5 The objective Hessian

Computing the objective Hessian is commonly one of the heaviest task of optimization algorithms. RMTR thus features a flexible strategy that allows the user to approximate it using the LTS package (implementing the lower-triangular substitution (LTS) method of Powell and Toint, 1979).

If it is easy to compute the objective Hessian, the user may provide a routine `HESS` that computes the objective Hessian and stores it in a sparse way. The header of this routine has to be:

```
SUBROUTINE HESS(x, Hval, Hrow, Hcol, Hnz)
```

where

`x` is an `INTENT(IN)` array of type `REAL` (double precision in `GALAHAD_RMTR_double`) that holds the point where the objective Hessian is computed.

`Hval, Hrow, Hcol` are pointers of type `REAL` (double precision in `GALAHAD_RMTR_-double`), default `INTEGER` and default `INTEGER`, respectively. They hold the objective Hessian stored using a sparse storage type (either COO or CSR format, depending on `control%matrix_storage`). Note that these are pointers since it may be necessary to deallocate them and then to allocate them with an appropriate size.

`Hnz` is an `INTENT(OUT)` scalar of type default `INTEGER` that holds the number of nonzero elements in the objective Hessian.

On the other hand, if it is not easy to compute the Hessian, the algorithm may call the LTS package to obtain a Hessian approximation. The user must in this case provide the sparsity structure of the Hessian. This may be done in several ways:

1. A function $f : \mathbb{R}^n \to \mathbb{R}$ is partially separable whenever there exists a decomposition of the form

$$f(x) = \sum_{k=1}^{k_{\max}} f_k(x), \tag{D.1}$$

where each element function $f_k : \mathbb{R}^n \Rightarrow \mathbb{R}$ $(k = 1, \ldots, k_{\max})$ depends only on a few variables, say those in some subset $I_k \subset \{1, \ldots, n\}$ with $|I_k| = n_k \ll n$. A lot of multilevel problems, especially in the context of discretization problems have a partially separable structure. The RMTR package features the use of such functions and the user has only to provide the partial separability decomposition of the Hessian with the routine `STRUCT`:

```
SUBROUTINE STRUCT(nbvar, nbrelem, xelvar, elvar)
```

where

`nbvar` is an `INTENT(IN)` scalar of type default `INTEGER` that holds the size of the problem variables at the current level.

`nbrelem` is an `INTENT(OUT)` scalar of type default `INTEGER` that holds the number of element functions $k_{\max}$ in the partial separability definition of the objective function.

`xelvar, elvar` are pointers of type default `INTEGER` that hold the partial separability structure of the objective Hessian in a framework related to the CSR storage for sparse matrices. The l-th entry of the vector `xelvar` contains the index of the first component in the vector `elvar` where an element of the l-th function is stored, and the vector `elvar` contains the list of the variables indices used by each separable function. The last element of `xelvar` is the size of `elvar` plus one.

2. The user may provide directly the Hessian sparsity pattern with the routine `SPARSITY`:

```
SUBROUTINE SPARSITY(nbvar, Hnz, srow, scol)
```

   where

   `nbvar` is an `INTENT(IN)` scalar of type default `INTEGER` that holds the size of the problem variables at the current level.

   `Hnz` is an `INTENT(OUT)` scalar of type default `INTEGER` that holds the number of nonzero elements in the objective Hessian.

   `srow, scol` are pointers of type default `INTEGER` that hold the sparsity pattern of the objective Hessian, meaning that they corresponds to the row and column arrays of the CSR storage of the Hessian.

3. Finally, the user may use one of the predefined sparsity patterns in the LTS package, by setting the `predefined-sparsity-pattern` option (see the LTS package documentation for a list of allowed values and a description of the corresponding sparsity patterns).

Note that the user only needs either to provide one of the three routines `HESS`, `STRUCT` or `SPARSITY`, or to specify a predefined sparsity pattern to use. The choice between these four possibilities has to be made by specifying the `approximate-Hessian` parameter in the problem specification file.

D.2.5.6 The bound constraints

The user may also specify bound constraints on the problem by providing the routines `LOWER_BOUND` and/or `UPPER_BOUND`:

```
SUBROUTINE LOWER_BOUND(lb)
SUBROUTINE UPPER_BOUND(ub)
```

where

`lb` is an `INTENT(INOUT)` array of type default `REAL` that holds the lower-bound constraints at the finest level, and

`ub` is an `INTENT(INOUT)` array of type default `REAL` that holds the upper-bound constraints at the finest level.

Note that it is not necessary to construct these subroutines if the problem admits no bound constraints. The lower- and upper-bound constraints are considered only if the parameters `lower-bound` and `upper-bound`, respectively, are set to `.TRUE.` in the problem specification file.

D.2.5.7 Transfer operators and number of variables

The RMTR algorithm needs information on the multilevel structure of the problem. The difficulty herein is that multilevel problems may be quite different, even in the restricted class of discretized optimization problems, depending for instance on the chosen discretization technique and domain topology. But, on the other hand, many common multilevel problems arise from discretizations of an underlying infinite-dimensional problem on regular geometric grids. To cover as much ground as possible with the RMTR package, we have therefore chosen a flexible two-pronged strategy: the user may either provide its own multilevel information, or use the package facilities for problems defined on some particular regular discretization grids.

If the predefined operators are not used (that is when `operators_type = GALAHAD_USER`), the user has to provide a routine that gives the number of the variables at each level, and whose header is

```
SUBROUTINE SIZES(sizes)
```

where

`sizes` is an `INTENT(INOUT)` array of type default `INTEGER`, whose length is the number of levels, and whose i-th entry holds the number of variables at level i.

He also has to specify his own prolongation and restriction operators with two routines whose headers are

```
SUBROUTINE PROLONGATION(P_val, P_row, P_col, p, q, nbvar)
SUBROUTINE RESTRICTION (R_val, R_row, R_col, p, q, nbvar, sigma)
```

where

`P_val, P_row, P_col` are pointers of type `REAL` (double precision in `GALAHAD_RMTR_double`), default `INTEGER` and default `INTEGER` , respectively. They hold the prolongation operator from level i to the finer level $i+1$ stored using a sparse storage (either COO or CSR, depending on `matrix_storage`).

`R_val, R_row, R_col` are pointers of type `REAL` (double precision in `GALAHAD_RMTR_double`), default `INTEGER` and default `INTEGER` , respectively. They hold the restriction operator from level i to the coarser level $i-1$ stored using a sparse storage (either COO or CSR, depending on `matrix_storage`).

`p, q` are `INTENT(OUT)` scalars of type default `INTEGER` that hold the number of lines and columns, respectively, of the operator.

`nbvar` is an `INTENT(IN)` scalar of type default `INTEGER` that holds the size of the variables of level i.

`sigma` is an `INTENT(OUT)` scalar of type `REAL` (double precision in `GALAHAD_RMTR_double`) that holds the 1-norm of the restriction operator from level i to level $i-1$.

Other choices of parameter `operators_type` indicate the use of predefined operators for geometric grids. The prolongation operator P_i (from level $i-1$ to level i) is then defined as the Kronecker product of unidimensional interpolation operators, while the corresponding restriction operator (from level i to level $i-1$) is defined from relation $R_i = P_i^T \|P_i\|_1^{-1}$. The selection of the grid hierarchy type is controlled by the following parameters:

`problem_dimension` is a scalar variable of type default `INTEGER`, that holds the problem dimension. This variable does not correspond to the dimension of the variable x, but to the dimension of the geometric space on which the problem is posed. The default is `problem_dimension = 2`.

`number_of_field_variables` is a scalar variable of type default `INTEGER`, that holds the number of variable fields of the problem, that is the number of variables that are defined at each node of the grid. The default is `number_of_field_variables = 1`.

`operators_type` determines the order of interpolation to define the prolongation operators (see the former description of the values allowed for this parameter).

`border_type` is a rank-one array variable of symbolic values, whose length is equal to `problem_dimension`, and whose j-th entry indicates the type of boundary condition along the j-th dimension of the grid. These entries may take the values `OPERATORS_INTERIOR`, `OPERATORS_EXTERIOR`, `OPERATORS_LEFT`, `OPERATORS_RIGHT`, `OPERATORS_LEFT_PERIODIC`, or `OPERATORS_RIGHT_PERIODIC`. We refer to Section 8.1 for a description of these options.

`coarsest_mesh_size` is a rank-one array variable of type default `INTEGER`, whose length is equal to `problem_dimension`, and whose j-th entry holds the number of variables in the j-th dimension of the grid.

The last two parameters must be provided as arguments of the `RMTR_initialize` routine. The number of variables at a given level is the product of `number_of_field_variables` by the number of variables in each dimension of the grid. The number of variables in the j-th dimension of the grid at every level i is recurred from the j-th entry of `coarsest_mesh_size` using the rule defined by the j-th entry of `border_type`.

D.2.6 Warning and error messages

A negative value of `inform%status` on exit from `RMTR_initialize`, `RMTR_solve` or `RMTR_terminate` indicates that an error has occurred. No further calls

should be made to these routines until the error has been corrected. Possible values are:

`-1` The memory allocation failed.

`-2` A file cannot be opened.

`-3` Impossible to write into a file.

`-4` Impossible to read a file.

`-6` Wrong input (negative level for example).

`-7` A vector has a wrong size.

`-9` An attempt to restrict a vector from the coarsest level was made.

`-10` An attempt to prolongate a vector from the finest level was made.

`-21` An IO error occurred while saving checkpointing information on the relevant disk file.

`-23` An input is not associated.

`-29` `inform%status` is not correct.

`-30` The maximum number of iterations has been reached and computation terminated.

`-31` Further progress of the algorithms appears to be impossible, although successful termination is not recognized.

Whatever the error status is, more information could be obtained by printing `inform%message`. Note that when the value of a parameter in a specification file is not correct, a warning message is printed though the algorithm still continues with the default parameter value replacing the wrong value.

D.2.7 The control and problem specification files

In this section, an alternative way of setting control parameters is described, that is components of the variable `control` of type `RMTR_control_type` (see Section D.2.3.2), by reading an appropriate data specification file. This facility is useful as it allows a user to change RMTR control parameters without editing and recompiling programs that call RMTR.

A specification file, or specfile, is a data file containing a number of *specification commands*. Each command occurs on a separate line, and comprises a *keyword*, which is a string (in a close-to-natural language) used to identify a control parameter, and an (optional) *value*, which defines the value to be

assigned to the given control parameter. All keywords and values are case insensitive, keywords may be preceded by one or more blanks but values must not contain blanks, and each value must be separated from its keyword by at least one blank. Values must not contain more than 30 characters, and each line of the specfile is limited to 80 characters, including the blanks separating keyword and value.

The keywords that are used by the RMTR package are simply the name of the components in the `RMTR_control_type` structure, whose underscores are replaced by hyphens. For example, the keyword `print-level` may used to change the component `print_level` in the control structure.

RMTR uses two different specification files. The first one is the control specification file that contains all the algorithmic parameters. The second one is the problem specification file that contains the parameters defining the problem. The portion of the specification file used by RMTR must start with a `BEGIN xxx` command, where `xxx` is either `RMTR` for the control specification file or `PROBLEM` for the problem specification file, and ends with an `END` command. The syntax of the specification files is thus defined as follows:

```
( ... lines ignored ... )
  BEGIN xxx
  keyword-1         value-1
     ...              ...
  keyword-n         value-n
  END
( ... lines ignored ... )
```

where `keyword-i` and `value-i` are strings separated by (at least) one blank. The `BEGIN xxx` and `END` delimiter command lines may contain additional (trailing) strings so long as such strings are separated by one or more blanks. For instance lines such as

```
  BEGIN xxx SPECIFICATION
```

and

```
  END xxx SPECIFICATION
```

are acceptable. Furthermore, between the `BEGIN xxx` and `END` delimiters, specification commands may occur in any order. Blank lines and lines whose first non-blank character is `!` or `*` are ignored. The content of a line after a `!` or `*` character is also ignored (as is the `!` or `*` character itself). This provides an easy manner to comment off some specification commands, or to comment specific values of certain control parameters.

The value of a control parameters may be of five different types, namely integer, logical, real, string or symbol. Integer and real values may be expressed in any relevant Fortran integer and floating-point formats (respectively). Permitted values for logical parameters are `ON`, `TRUE`, `.TRUE.`, `T`, `YES`, `Y`, or `OFF`, `NO`, `N`, `FALSE`, `.FALSE.`, `F`. Empty values are also allowed for logical control parameters, and are interpreted as `TRUE`. String are specified as a sequence

of characters. A symbolic value is a special string obtained from one of the predefined symbols of the `SYMBOLS` module by deleting the leading '`GALAHAD_`' characters in its name. Thus, the specification command

```
print-level SILENT
```

implies that the value `GALAHAD_SILENT` is assigned to `control%print-level`. This technique is intended to help expressing an (integer) control parameter for an algorithm in a 'language' that is close to natural. A default specification file is in the `src` directory. The complete list of parameters is enclosed in Table D.1 (control parameters) and Table D.2 (problem parameters) with their types and default values. Note that all these parameters are stored in the `RMTR_control_type` structure described in Subsection D.2.3.2 and that their name in the structure is simply the name of the parameter where the hyphen signs are replaced by underscore signs.

Name of the parameter	Value type / Symbolic value	Default value
`error-printout-device`	`INTEGER`	6
`printout-device`	`INTEGER`	6
`print-level`	`SILENT`, `SUMMARY`, `TRACE`, `ACTION`, `DETAILS`, `DEBUG`, `CRAZY`	`TRACE`
`start-printing-at-iteration`	`INTEGER`	0
`stop-printing-at-iteration`	`INTEGER`	-1
`display-equivalent-evaluations`	`LOGICAL`	`.TRUE.`
`display-options`	`LOGICAL`	`.TRUE.`
`save-solution`	`LOGICAL`	`.TRUE.`
`criticality-threshold`	`REAL`	10^{-6}
`function-threshold`	`REAL`	10^{20}
`truncated-conjugate-gradient-accuracy`	`REAL`	10^{-1}
`maximum-number-of-iterations`	`INTEGER`	1000
`maximum-number-of-tcg-iterations`	`INTEGER`	5
`maximum-solving-time`	`REAL`	3600.0
`minimum-rho-for-successful-iteration`	`REAL`	0.01
`minimum-rho-for-very-successful-iteration`	`REAL`	0.9
`radius-reduction-factor`	`REAL`	0.25
`radius-increase-factor`	`REAL`	2.0
`maximum-radius-increase-factor`	`REAL`	3.0
`maximum-radius`	`REAL`	-1.0
`initial-radius`	`REAL`	1.0
`forced-Hessian-estimation-frequency`	`INTEGER`	0
`forced-Hessian-evaluation-factor`	`REAL`	0.5
`euclidean-gradient-accuracy-for-Hessian-evaluation`	`REAL`	0.15
`infinite-gradient-accuracy-for-Hessian-evaluation`	`REAL`	10000.0
`initialization-technique`	`AF`, `MR`,	

continued on next page...

Name of the parameter	Value type / Symbolic value	Default value
	FM, MF, FMF	FM
cycling-style	Vcycles, Wcycles, freecycles	Vcycles
coarse-model-choice-parameter	REAL	0.25
linesearch	INTEGER	2
model-backtracking	LOGICAL	.TRUE.
quadratic-model	FIRST_ORDER, SECOND_ORDER, GALERKIN	GALERKIN
number-of-smoothing-cycles	INTEGER	7
smooth-frequency	NEVER_SMOOTH, SMOOTH_DOWN, SMOOTH_UP, ALWAYS_SMOOTH	ALWAYS_SMOOTH
checkpointing-frequency	INTEGER	0
checkpointing-file	CHARACTER	RMTR.sav
checkpointing-device	INTEGER	55
restart-from-checkpoint	LOGICAL	.FALSE.

Table D.1 — *Control parameters with their types and default values.*

Name of the parameter	Value type / Symbolic value	Default value
problem-dimension	INTEGER	2
level-min	INTEGER	1
level-max	INTEGER	4
operators-type	USER, LINEAR, LINEAR_CUBIC, CUBIC	LINEAR_CUBIC
matrix-storage	COORDINATE, SPARSE_BY_ROWS	COORDINATE
half-Hessian	LOGICAL	.FALSE.
number-of-field-variables	INTEGER	1
upper-bound	LOGICAL	.FALSE.
lower-bound	LOGICAL	.FALSE.
quadratic-problem	LOGICAL	.FALSE.
starting-point-file	CHARACTER	RMTR_starting-point.dat
solution-file	CHARACTER	RMTR_solution.dat
approximate-Hessian	EXACT_HESSIAN, LTS_STRUCT, LTS_SPARSITY, LTS_PREDEFINED-_PATTERN	EXACT_HESSIAN
predefined-sparsity-pattern	INTEGER	0

Table D.2 — *Problem parameters with their types and default values.*

D.2.8 Information printed

The meaning of the various `control%print_level` values is defined as follows:

`GALAHAD_SILENT:` nothing is printed.

`GALAHAD_SUMMARY:` only reports a summary of the iterations at the end of the algorithm.

`GALAHAD_TRACE:` reports a one line summary of each iteration. This summary includes the current level index, the number of variables of the current level, the current number of iterations at this level, the current values of the objective function, the criticality measure value, the step norm, the trust-region radius, the ratio of achieved to predicted reduction, the iteration type, the model decrease and the objective function decrease. The iteration type is a six character string which can be

- `LOWER␣` if the algorithm just starts a recursion,
- `TAYLOR` if the algorithm minimizes the Taylor model using a standard trust-region technique,
- `SMOOTH` if the algorithm minimizes the Taylor model using a smoothing technique,
- `BACKTR` if the algorithm makes a backtracking iteration, or
- `UPPER␣` if the algorithm just finishes a recursion.

`GALAHAD_ACTION:` reports the mains steps of each iteration.

`GALAHAD_DETAILS, GALAHAD_DEBUG` and `GALAHAD_CRAZY` report more and more information.

Note that, after the summary of the iterations, the algorithm reports a summary of the equivalent iterations, that is the total amount of work expressed as work in the finest level (see Gratton *et al.*, 2010*b*). This measure is a better measure of the total amount of work done at all levels. The equivalent quantities reported are:

`f evaluations` corresponds to the equivalent number of objective function evaluations.

`g evaluations` corresponds to the equivalent number of objective gradient evaluations.

`H evaluations` corresponds to the equivalent number of objective Hessian evaluations

`smoothing cycles` corresponds to the equivalent number of SCM smoothing cycles.

`Taylor iterations` corresponds to the equivalent number of PTCG iterations.

`H updates` corresponds to the equivalent number of Hessian updates (in the case of approximate Hessians), that is, the number of times where the Hessian values are recomputed (by evaluating gradients, the structure is never recomputed).

`H eval+upd` corresponds to the sum of the equivalent number of Hessian evaluations and updates.

The algorithm finally reports the total amount of solving time and the total amount of time to solve the problem (including the time spent to construct the multilevel structure, the transfer operators, ...).

D.3 General information

Use of common: None.

Workspace: Provided automatically by the module.

Other routines called directly: `RMTR_solve` calls the BLAS routines `*NRM2` and `*DOT`, where * is `S` for the default real version and `D` for the double precision version. The routines `amubdg`, `amub`, `aplbdg`, `aplb`, `coocsr`, `csrcsc`, `csrssr`, and `ssrcsr` from the `SPARSKIT` library developed by Saad (1994) are also used.

Other modules used directly: `RMTR` calls GALAHAD modules `LTS`, `SPECFILE` and `SYMBOLS`

Input/output: Output is under control of the arguments `control%error_printout_device`, `control%printout_device` and `control%print_level`.

Restrictions: `problem%nbvar > 0`.

Portability: ISO Fortran 95 + TR 15581 or Fortran 2003. The package is thread-safe.

D.4 Example of use

Consider the minimum surface problem

$$\min_{v \in \mathcal{K}} \int_{S_2} \sqrt{1 + \|\nabla v\|_2^2},$$

where $\mathcal{K} = \left\{v \in H^1(S_2) \mid v(x) = v_0(x) \text{ on } \partial S_2\right\}$ and $S_2 = [0,1] \times [0,1]$. This convex problem is discretized using a finite-element basis defined using a uniform triangulation of S_2, with the same grid spacing, h, along the two coordinate directions. The basis functions are the classical P1 functions which are linear on each triangle and take the value 0 or 1 at each vertex. The boundary condition $v_0(x)$ is chosen as

$$v_0(x) = \begin{cases} f(x_1), & x_2 = 0, \quad 0 \leqslant x_1 \leqslant 1, \\ 0, & x_1 = 0, \quad 0 \leqslant x_2 \leqslant 1, \\ f(x_1), & x_2 = 1, \quad 0 \leqslant x_1 \leqslant 1, \\ 0, & x_1 = 1, \quad 0 \leqslant x_2 \leqslant 1, \end{cases}$$

where $f(x_1) = x_1(1 - x_1)$. The problem has the following lower bound constraint:

$$v(x) \geqslant \sqrt{2} \qquad \text{whenever} \quad \frac{4}{9} \leqslant x_1, x_2 \leqslant \frac{5}{9},$$

thereby creating an obstacle problem where the surface is constrained in the middle of the domain.

The call to the `RMTR` package may be done by

```
PROGRAM GALAHAD_RMTR_EXAMPLE

  ! RMTR module
  USE GALAHAD_RMTR_double

  ! GALAHAD symbols
  USE GALAHAD_SYMBOLS, OK => GALAHAD_SUCCESS

  IMPLICIT NONE

  ! RMTR data structures
  TYPE(RMTR_inform_type )           :: inform
  TYPE(RMTR_control_type)          :: control
  TYPE(RMTR_problem_type), POINTER :: problem

  ! Problem routines
  EXTERNAL :: cost, grad, Hessian, lower_bound

  ! Initialization
  ALLOCATE(problem)

  CALL RMTR_initialize(control, inform, problem, specname, &
       problemspecname, GRAD=grad, LOWER_BOUND=lower_bound)

  ! Solution
  IF(inform%status == OK)THEN
     CALL RMTR_solve(control, inform, problem, FUN=cost,   &
          GRAD=grad, HESS=Hessian)
  END IF

  ! Termination
  CALL RMTR_terminate(control, inform, problem)
```

```
  DEALLOCATE(problem)

END PROGRAM GALAHAD_RMTR_EXAMPLE
```

The routines `f0`, `f1`, `f2` and `f3` to compute the boundary conditions of the problem are given by

```
!=============================================================
REAL(KIND = wp) FUNCTION f0(x)

  ! Compute the south boundary of the problem

  INTEGER, PARAMETER :: wp = KIND(1.0D+0)
  REAL(KIND=wp)      :: x

  f0 = x*(1-x)

END FUNCTION f0
!=============================================================
REAL(KIND=wp) FUNCTION f1(x)

  ! Compute the west boundary of the problem

  INTEGER, PARAMETER :: wp = KIND(1.0D+0)
  REAL(KIND=wp)      :: x

  f1 = 0.0_wp

END FUNCTION f1
!=============================================================
REAL(KIND=wp) FUNCTION f2(x)

  ! Compute the north boundary of the problem

  INTEGER, PARAMETER :: wp = KIND(1.0D+0)
  REAL(KIND=wp)      :: x

  f2 = x*(1-x)

END FUNCTION f2
!=============================================================
REAL(KIND=wp) FUNCTION f3(x)

  ! Compute the east boundary of the problem

  INTEGER, PARAMETER :: wp = KIND(1.0D+0)
  REAL(KIND=wp)      :: x

  f3 = 0.0_wp

END FUNCTION f3
!=============================================================
```

The routines `cost`, `grad`, `Hessian` and `compute_lower_bound` which compute the objective function, gradient and Hessian and the lower bound of the problem, respectively are given by

```
!============================================================

SUBROUTINE cost(x,f)

! ** Objective function of the minimal surface problem **

  INTEGER, PARAMETER :: wp = KIND(1.0D+0)
  REAL(KIND=wp), INTENT(IN ), DIMENSION(:) :: x ! Iterate
  REAL(KIND=wp), INTENT(OUT)               :: f ! Cost

! Local variables
  INTEGER       :: i, j, temp
  REAL(KIND=wp) :: a, h, p, q, r, s, c, c1, c2, lc, k
  REAL(KIND=wp), ALLOCATABLE, DIMENSION(:,:) :: x_mod

! Compute the mesh size and the size of the grid
  a    = SIZE(x)
  a    = SQRT(a)
  temp = NINT(a)
  h    = 1.0_wp / (a+1.0_wp)

! Compute the grid with the boundary
  f = 0.0_wp
  ALLOCATE(x_mod(temp+2,temp+2))
  x_mod = 0.0_wp

  DO i = 1, temp
     x_mod(2:temp+1,i+1) = x(1+(i-1)*temp:i*temp)
  END DO

! Compute the boundary
  DO j = 1, temp+2
     k = REAL(j-1, KIND=wp)
     x_mod(1,j)      = f0(k/(temp+1))
     x_mod(j,1)      = f1(k/(temp+1))
     x_mod(temp+2,j) = f2(k/(temp+1))
     x_mod(j,temp+2) = f3(k/(temp+1))
  END DO

! Compute the objective function
  DO i = 1, temp+1
     DO j = 1, temp+1
        p  = x_mod(i,  j)
        q  = x_mod(i+1,j)
        r  = x_mod(i+1,j+1)
        s  = x_mod(i,  j+1)
        c1 = SQRT(1.0_wp + ((s-p)/h)**2 + ((s-r)/h)**2)
        c2 = SQRT(1.0_wp + ((p-q)/h)**2 + ((r-q)/h)**2)
        lc = 0.5_wp * (c1+c2) * h**2
        f  = f + lc
     END DO
```

```
  END DO
  DEALLOCATE(x_mod)

END SUBROUTINE cost

!=============================================================

SUBROUTINE grad(x, g)

! ** Objective gradient of the minimal surface problem **

  INTEGER, PARAMETER :: wp = KIND(1.0D+0)
  REAL(KIND=wp), INTENT(IN ), DIMENSION(:) :: x ! Iterate
  REAL(KIND=wp), INTENT(OUT), DIMENSION(:) :: g ! Gradient

! Local variables
  INTEGER       :: i, j, temp, pt1, pt2, pt3, pt4
  REAL(KIND=wp) :: a, h, p, q, r, s, c1, c2, lc, k
  REAL(KIND=wp), ALLOCATABLE, DIMENSION(:,:) :: x_mod
  REAL(KIND=wp), ALLOCATABLE, DIMENSION(:)   :: g_mod

! Compute the mesh size and the size of the grid
  a    = SQRT(SIZE(x))
  temp = NINT(a)
  h    = 1.0_wp / (a+1.0_wp)

! Compute the grid and gradient with the boundary components
  ALLOCATE(x_mod(temp+2,temp+2), g_mod((temp+2)**2)))
  x_mod = 0.0
  g_mod = 0.0
  DO i = 1, temp
     x_mod(2:temp+1,i+1) = x(1+(i-1)*temp:i*temp)
  END DO

! Compute the boundary
  DO j = 1, temp+2
     k = REAL(j-1, KIND=wp)
     x_mod(1,j)      = f0(k/(temp+1))
     x_mod(j,1)      = f1(k/(temp+1))
     x_mod(temp+2,j) = f2(k/(temp+1))
     x_mod(j,temp+2) = f3(k/(temp+1))
  END DO

! Compute the gradient
  DO i = 1, temp+1
     DO j = 1, temp+1
        ! Compute the neighbours indices in the gradient
        pt1  = (j-1) * (temp+2) + i
        pt2  = pt1+1
        pt3  = pt2 + (temp+2)
        pt4  = pt3-1

        ! Select the neighbours
        p=x_mod(i,j)
        q=x_mod(i+1,j)
```

```
        r=x_mod(i+1,j+1)
        s=x_mod(i,j+1)

        ! Compute the derivatives
        c1 = SQRT(1.0_wp + ((s-p)/h)**2 + ((s-r)/h)**2)
        c2 = SQRT(1.0_wp + ((p-q)/h)**2 + ((r-q)/h)**2)

        g_mod(pt1) = g_mod(pt1) + ((p-s)/c1 + (p-q)/c2)/2
        g_mod(pt2) = g_mod(pt2) + (2*q-p-r)/2/c2
        g_mod(pt3) = g_mod(pt3) + ((r-s)/2/c1 + (r-q)/2/c2)
        g_mod(pt4) = g_mod(pt4) + (2*s-p-r)/2/c1
     END DO
  END DO

  ! Select the free components
  DO i = 1, temp
     g(1+(i-1)*temp:i*temp) =                                &
          g_mod(2+(i)*(temp+2):(i+1)*(temp+2)-1)
  END DO

  DEALLOCATE(x_mod, g_mod)

END SUBROUTINE grad

!=============================================================

SUBROUTINE Hessian(x, Hval, Hrow, Hcol, Hnz)

! ** Objective Hessian of the minimal surface problem **

  INTEGER, PARAMETER :: wp = KIND(1.0D+0)
  REAL(KIND=wp), INTENT(IN), DIMENSION(:) :: x    ! Iterate
  REAL(KIND=wp), POINTER,    DIMENSION(:) :: Hval ! Hessian
  INTEGER,       POINTER,    DIMENSION(:) :: Hrow !  in CSR
  INTEGER,       POINTER,    DIMENSION(:) :: Hcol !  storage
  INTEGER,       INTENT(OUT)              :: Hnz  ! Hess nnz

! Local variables
  REAL(KIND=wp) :: a, h, p, q, r, s, c1, c2, k
  REAL(KIND=wp) :: xmt, zmt, xmy, zmy, dtmxmz, dymxmz
  INTEGER :: i, j, temp, ind, tmp_Hnz, pt1, pt2, pt3, pt4, col, &
      colm, row, rowm
  REAL(KIND=wp), ALLOCATABLE, DIMENSION(:,:) :: x_mod
  REAL(KIND=wp), ALLOCATABLE, DIMENSION(:)   :: g1, g2, DD0, DD1, &
      DD2, DD3
  REAL(KIND=wp), ALLOCATABLE, DIMENSION(:)   :: tmp_Hval
  INTEGER,       ALLOCATABLE, DIMENSION(:)   :: tmp_Hrow
  INTEGER,       ALLOCATABLE, DIMENSION(:)   :: tmp_Hcol

! Compute the mesh size and the size of the grid
  a    = SQRT(SIZE(x))
  temp = NINT(a)
  h    = 1.0_wp / (a+1.0_wp)

! Compute the grid with the boundary
```

```
  ALLOCATE(x_mod(temp+2,temp+2))
  x_mod = 0.0_wp
  DO i = 1, temp
     x_mod(2:temp+1,i+1) = x(1+(i-1)*temp:i*temp)
  END DO

! Compute the boundary
  DO j = 1, temp+2
     k = REAL(j-1, KIND=wp)
     x_mod(1,j)      = f0(k/(temp+1.0))
     x_mod(j,1)      = f1(k/(temp+1.0))
     x_mod(temp+2,j) = f2(k/(temp+1.0))
     x_mod(j,temp+2) = f3(k/(temp+1.0))
  END DO

! Allocate local variables and the Hessian bands
  ALLOCATE(g1(4), g2(4))
  ALLOCATE(DD0((temp+2)*(temp+2)),  DD1(SIZE(DD0)-1)       )
  ALLOCATE(DD2(SIZE(DD0)-(temp+2)), DD3(SIZE(DD0)-(temp+3)))
  DD0 = 0.0_wp
  DD1 = 0.0_wp
  DD2 = 0.0_wp
  DD3 = 0.0_wp

! Compute the Hessian
  DO i = 1, temp+1
     DO j = 1, temp+1
        ! Compute the neighbours indices
        pt1  = (j-1) * (temp+2) + i
        pt2  = pt1+1
        pt3  = pt2 + (temp+2)
        pt4  = pt3-1

        ! Select the neighbours
        p=x_mod(i,j)
        q=x_mod(i+1,j)
        r=x_mod(i+1,j+1)
        s=x_mod(i,j+1)

        ! Compute the derivatives
        c1 = SQRT(1.0_wp+ ((s-p)/h)**2 + ((s-r)/h)**2)
        c2 = SQRT(1.0_wp+ ((p-q)/h)**2 + ((r-q)/h)**2)

        g1(1) = 1.0_wp/(h**2)/c1*(p-s)
        g1(2) = 0.0_wp
        g1(3) = 1.0_wp/(h**2)/c1*(r-s)
        g1(4) = 1.0_wp/(h**2)/c1*(2*s-p-r)
        g2(1) = 1.0_wp/(h**2)/c2*(p-q)
        g2(2) = 1.0_wp/(h**2)/c2*(2*q-p-r)
        g2(3) = 1.0_wp/(h**2)/c2*(r-q)
        g2(4) = 0.0_wp

        xmt    = p-s
        zmt    = r-s
        dtmxmz = 2*s-p-r
```

```
        xmy    = p-q
        zmy    = r-q
        dymxmz = 2*q-p-r

        ! Compute each band of the Hessian
        DD0(pt1) = DD0(pt1)                                          &
             + c1/(c1**2)/2-g1(1)*xmt/(c1**2)/2                      &
             + c2/(c2**2)/2-g2(1)*xmy/(c2**2)/2
        DD0(pt2) = DD0(pt2)                                          &
             + 2*c2/(c2**2)/2-g2(2)*dymxmz/(c2**2)/2
        DD0(pt3) = DD0(pt3)                                          &
             + c1/(c1**2)/2-g1(3)*zmt/(c1**2)/2                      &
             + c2/(c2**2)/2-g2(3)*zmy/(c2**2)/2
        DD0(pt4) = DD0(pt4)                                          &
             + 2*c1/(c1**2)/2-g1(4)*dtmxmz/(c1**2)/2
        DD1(pt1) = DD1(pt1)                                          &
             - g1(2)*xmt/(c1**2)/2-c2/(c2**2)/2                      &
             - g2(2)*xmy/(c2**2)/2
        DD2(pt1) = DD2(pt1)                                          &
             - c1/(c1**2)/2-g1(4)*xmt/(c1**2)/2                      &
             - g2(4)*xmy/(c2**2)/2
        DD3(pt1) = DD3(pt1)                                          &
             - g1(3)*xmt/(c1**2)/2-g2(3)*xmy/(c2**2)/2
        DD2(pt2) = DD2(pt2)                                          &
             - c2/(c2**2)/2-g2(3)*dymxmz/(c2**2)/2
        DD1(pt4) = DD1(pt4)                                          &
             - c1/(c1**2)/2-g1(4)*zmt/(c1**2)/2                      &
             -g2(4)*zmy/(c2**2)/2
     END DO
  END DO
  DEALLOCATE(g1, g2, x_mod)

! Compute the number of nonzero entries in the Hessian
  tmp_Hnz = 2*(SIZE(DD3)+SIZE(DD2)+SIZE(DD1)) + SIZE(DD0)

  IF(ASSOCIATED(Hval)) DEALLOCATE(Hval)
  IF(ASSOCIATED(Hrow)) DEALLOCATE(Hrow)
  IF(ASSOCIATED(Hcol)) DEALLOCATE(Hcol)

! Compute the Hessian with the boundary elements
  ALLOCATE(tmp_Hval(tmp_Hnz))
  ALLOCATE(tmp_Hrow(tmp_Hnz))
  ALLOCATE(tmp_Hcol(tmp_Hnz))

! Add each band
  Hnz = 0
  ind = 1
  DO i = 1, SIZE(DD3)
     tmp_Hval(ind) = DD3(i)
     tmp_Hrow(ind) = i
     tmp_Hcol(ind) = (temp+3)+i
     ind = ind +1
     tmp_Hval(ind) = DD3(i)
     tmp_Hrow(ind) = (temp+3)+i
     tmp_Hcol(ind) = i
```

```
     ind = ind +1
  END DO
  DO i = 1, SIZE(DD0)
     tmp_Hval(ind) = DD0(i)
     tmp_Hrow(ind) = i
     tmp_Hcol(ind) = i
     ind = ind +1
  END DO
  DO i = 1, SIZE(DD1)
     tmp_Hval(ind) = DD1(i)
     tmp_Hrow(ind) = i
     tmp_Hcol(ind) = i+1
     ind = ind +1
     tmp_Hval(ind) = DD1(i)
     tmp_Hrow(ind) = i+1
     tmp_Hcol(ind) = i
     ind = ind +1
  END DO
  DO i = 1, SIZE(DD2)
     tmp_Hval(ind) = DD2(i)
     tmp_Hrow(ind) = i
     tmp_Hcol(ind) = (temp+2)+i
     ind = ind +1
     tmp_Hval(ind) = DD2(i)
     tmp_Hrow(ind) = (temp+2)+i
     tmp_Hcol(ind) = i
     ind = ind +1
  END DO
  DEALLOCATE(DD0, DD1, DD2, DD3)

! Compute the number of nonzero elements in the Hessian
! without the boundary
  Hnz = 0
  DO i = 1, SIZE(tmp_Hval)
     row  = tmp_Hrow(i)
     col  = tmp_Hcol(i)
     rowm = MOD(row,temp+2)
     colm = MOD(col,temp+2)
     IF ( (1 < rowm) .AND. (temp+2 > rowm) .AND.           &
          (0 < (row-rowm)/(temp+2))        .AND.           &
          (temp+1 > (row-rowm)/(temp+2))       ) THEN
        IF ( (1 < colm) .AND. (temp+2 > colm) .AND.        &
             (0 < (col-colm)/(temp+2))        .AND.        &
             (temp+1 > (col-colm)/(temp+2))       ) THEN
           Hnz = Hnz + 1
        END IF
     END IF
  END DO

! Compute the Hessian without the boundary
  ALLOCATE(Hval(Hnz), Hrow(Hnz), Hcol(Hnz))
  ind = 1
  DO i = 1, SIZE(tmp_Hval)
     row  = tmp_Hrow(i)
     col  = tmp_Hcol(i)
```

```
     rowm = MOD(row,temp+2)
     colm = MOD(col,temp+2)
     IF ( (1 < rowm) .AND. (temp+2 > rowm) .AND.           &
          (0 < (row-rowm)/(temp+2))        .AND.           &
          (temp+1 > (row-rowm)/(temp+2))        ) THEN
        IF ( (1 < colm) .AND. (temp+2 > colm) .AND.        &
             (0 < (col-colm)/(temp+2))        .AND.        &
             (temp+1 > (col-colm)/(temp+2))        ) THEN
           Hval(ind) = tmp_Hval(i)
           Hrow(ind) = row-(temp+2)-1-&
                2*(((row-rowm)/(temp+2))-1)
           Hcol(ind) = col-(temp+2)-1-            &
                2*(((col-colm)/(temp+2))-1)
           ind=ind+1
        END IF
     END IF
  END DO
  DEALLOCATE(tmp_Hval, tmp_Hrow, tmp_Hcol)

END SUBROUTINE Hessian

!=============================================================

SUBROUTINE compute_lower_bound(lb)

! ** Compute the lower bound on the problem s variables **

  REAL(KIND=wp), INTENT(INOUT), DIMENSION(:) :: lb

! Local variables
  INTEGER       :: i, j, k, temp, temp2
  REAL(KIND=wp) :: a
  REAL(KIND=wp), ALLOCATABLE,   DIMENSION(:) :: tmp

! Compute the size of the grid
  lb = 0.0_wp
  temp = size(lb)
  a = SQRT(DBLE(temp))
  temp = NINT(a)

! Compute the lower bound
  ALLOCATE(tmp(temp))
  tmp = 0.0_wp
  temp2 = (temp-MODULO(temp,9)) / 9
  IF(temp2>1) THEN
     i = 4*temp2 - 1
     DO WHILE(i<5*temp2)
        i=i+1
        tmp(i) = SQRT(2.0_wp)
     END DO
  END IF
  k = 1
  DO i = 1, SIZE(tmp)
     DO j = 1, SIZE(tmp)
        lb(k) = tmp(i)*tmp(j)
```

```
        k = k + 1
      END DO
  END DO
  DEALLOCATE(tmp)

END SUBROUTINE compute_lower_bound

!=============================================================
```

This example is the one implemented in the `rmtrs` program. The RMTR algorithm is launched with the following control and problem specification files:

```
BEGIN RMTR
error-printout-device                                    6
printout-device                                          6
print-level                                        SUMMARY
criticality-threshold                                0.001
truncated-conjugate-gradient-accuracy                  0.1
maximum-number-of-iterations                          1000
maximum-number-of-tcg-iterations                         5
initialization-technique                                FM
display-equivalent-evaluations                           T
cycling-style                                      Vcycles
minimum-rho-for-successful-iteration                  0.01
minimum-rho-for-very-successful-iteration               0.9
radius-reduction-factor                                0.25
radius-increase-factor                                  2.0
initial-radius                                          1.0
coarse-model-choice-parameter                          0.25
linesearch                                               2
model-backtracking                                       T
quadratic-model                                   GALERKIN
forced-Hessian-estimation-frequency                      0
forced-Hessian-evaluation-factor                       0.5
euclidean-gradient-accuracy-for-Hessian-evaluation    0.15
infinite-gradient-accuracy-for-Hessian-evaluation  10000.0
number-of-smoothing-cycles                               7
smooth-frequency                             ALWAYS_SMOOTH
start-printing-at-iteration                              0
stop-printing-at-iteration                              -1
maximum-radius                                     1.0D20
maximum-radius-increase-factor                          3.0
save-solution                                            T
display-options                                          F
checkpointing-frequency                                  0
checkpointing-file                                RMTR.sav
checkpointing-device                                    55
restart-from-checkpoint                                  F
maximum-solving-time                                    -1
END RMTR
```

```
BEGIN PROBLEM
problem-dimension           2
level-min                   1
level-max                   7
```

```
interpolation-type           2
approximate-Hessian          EXACT_HESSIAN
upper-bound                  F
lower-bound                  T
predefined-sparsity-pattern 1
starting-point-file          RMTR_startingpoint.dat
solution-file                RMTR_solution.dat
END PROBLEM
```

Here is the output for the problem.

```
Iterations summary:
-------------------

Level                :     1    2    3    4    5    6    7
-------------------------------------------------------------
Taylor               :   121    0    0    0    0    0    0
Taylor iterations    :   613    0    0    0    0    0    0
Smoothing            :     0  502  580  472  276  193  161
Cycles               :     0  544  623  472  276  193  161
Model-Backtracking   :     0    0    2    0   17   21   19
f evaluations        :     7    7   13  229  227  164  250
g evaluations        :     7    7   11  229  210  143  231
H evaluations        :     6    4    8    3   18   22   18
H updates            :     0    0    0    0    0    0    0
H reductions         :     0    6   25   23   23   14    6
Prolongations        :   115  290  291  179   82   69    0
Restrictions         :     0 1075  765 1751 1131  657  779
Projections          :     0    0    0    0    0    0    0
Rejected projections:     0    0    0    0    0    0    0

Equivalent evaluations number
-----------------------------
f evaluations      :   308.8250
g evaluations      :   283.5046
H evaluations      :    24.7085
smoothing cycles   :   236.8398
Taylor iterations  :     0.1497
H updates          :     0.0000
H eval+upd         :    24.7085

Total CPU time     :     24.922 second(s)
Solving time       :     24.722 second(s)

********************** Bye **********************
```

Appendix E

Specification of the LTS package

E.1 Summary

ATTRIBUTES — **Versions:** `GALAHAD_LTS_single`, `GALAHAD_LTS_double`. **Uses:** *COPY (from BLAS library). **Date:** June 2010. **Origin:** V. Malmedy, University of Namur (FUNDP) & Fonds de la Recherche Scientifique (FNRS). **Language:** Fortran 95 + TR 15581 or Fortran 2003.

E.2 How to use the package

Access to the package requires a `USE` statement such as
Single precision version:

```
USE GALAHAD_LTS_single
```

Double precision version:

```
USE GALAHAD_LTS_double
```

If it is required to use both modules at the same time, the derived type `LTS_col_groups` and the subroutines `LTS_powetoin`, `LTS_cpr_group`, `LTS_permute`, `LTS_partSepa2sparse`, `LTS_lowerpart`, `LTS_getOptimalGroups` must be renamed on one of the USE statements.

E.2.1 Matrix storage formats

As the LTS code aims to deal with large-scale problems, the matrices must be stored in a sparse format. We use several variants to benefit from their

respective advantages: the sparse coordinate (COO) format, the compressed sparse row (CSR) format and the compressed sparse column (CSC) format. In both cases, only the nonzero entries of the matrices are stored.

The COO format uses two integer arrays and a real array, whose size is the number of nonzero entries in the matrix. The ℓ-th entries of the two integers arrays holds the row and column index, respectively, of some nonzero entry of the matrix, whose value is stored in the ℓ-th entry of the real array. The order of the matrix entries is unimportant.

The CSR format also uses three arrays: an integer and a real arrays whose size is the number of nonzero entries in the matrix, as well as a second integer array whose size is the number of rows plus one. The first two arrays still holds the column index and value, respectively, of the matrix nonzero entries, but which are sorted by increasing row index. The second integer array holds, at place i, the index of the first entry in the other two arrays that corresponds to a matrix entry from the i-th row; its last entry holds the number of nonzero entries of the matrix plus one.

The CSC format differs from the CSR format only by the fact that it interchanges the role of rows and columns. This means that the CSR and CSC formats coincide for symmetric matrices, while the CSR storage of a non symmetric matrix constitutes a CSC storage for its transpose.

Conversions between sparse matrix formats (for instance, from COO to CSR format) may be performed by the appropriate subroutines from the SPARSEKIT library by Saad (1994).

E.2.2 The derived data type

A derived data type is accessible from the package, namely `LTS_col_groups`. It is used to store the information on the structure of the Hessian that is needed for its approximation. The components of `LTS_col_groups` are:

`p` is a scalar variable of type default `INTEGER`, that gives the number of column groups that are used for the Hessian approximation.

`prow` is an array variable of dimension `p` + 1 and type default `INTEGER`, that is used with variable `pcol` to store the column groups in a CSR way. It references the first column of each group.

`pcol` is an array variable of dimension n and type default `INTEGER`, that is used with variable `prow` to store the column groups in a CSR way. It holds the column indices gathered by group (in compressed sparse row format)

`pinv` is an array variable of dimension n and type default `INTEGER`, that indicates the group to which each column belongs.

`u` is an array variable of dimension n and type default `INTEGER`, that gives the symmetric permutation used on the Hessian structure to reduce the

number of gradient evaluations required for its approximation ($H(i,j) = H_{\text{permuted}}(u(i), u(j))$).

`w` is an array variable of dimension n and type default `INTEGER`, that gives the inverse symmetric permutation ($H(w(i), w(j)) = H_{\text{permuted}}(i,j)$).

`lrow` is an array variable of type default `INTEGER`, that is used with variable `lcol` to store the sparsity structure of the triangular lower part of the Hessian in a COO way. It holds the row indices of the matrix entries.

`lcol` is an array variable of type default `INTEGER`, that is used with variable `lrow` to store the sparsity structure of the triangular lower part of the Hessian in a COO way. It holds the column indices of the matrix entries.

E.2.3 Argument lists and calling sequences

There are six procedures for user calls:

1. The subroutine `LTS_powetoin` is the main subroutine and is used to approximate the Hessian.
2. The subroutine `LTS_cpr_group` is used to determine the column groups.
3. The subroutine `LTS_permute` is used to compute a symmetric permutation of rows and columns of the Hessian matrix that decreases the number of gradient evaluations required for its approximation.
4. The subroutine `LTS_partSepa2sparse` is used to convert a partial separability structure into a sparsity structure.
5. The subroutine `LTS_lowerpart` is used to obtain the lower triangular part of a sparsity structure.
6. The subroutine `LTS_getOptimalGroups` is used to obtain a predefined Hessian structure.

E.2.3.1 The Hessian approximation subroutine

The LTS algorithm for Hessian approximation is called as follows:

```
CALL LTS_powetoin(n, x, Hnz, Hval, Hrow, Hcol, group, permuted, &
    GRAD)
```

`n` is a scalar `INTENT(IN)` argument of type default `INTEGER`, that gives the problem size.

`x` is an array `INTENT(IN)` argument of dimension `n` and type default `REAL` (double precision in `GALAHAD_LTS_double`), that holds the point at which an approximation of the Hessian is computed.

`Hnz` is a scalar `INTENT(IN)` argument of type default `INTEGER`, that gives the number of possibly nonzero entries in the Hessian matrix.

`Hval` is an array `POINTER` argument of dimension `Hnz` and type default `REAL` (double precision in `GALAHAD_LTS_double`). On exit, `Hval` contains the values of the possibly nonzero entries of the approximated Hessian.

`Hrow` is an array `POINTER` argument of dimension `Hnz` and type default `INTEGER`. On exit, `Hrow` contains the row indices of the possibly nonzero entries of the approximated Hessian.

`Hcol` is an array `POINTER` argument of dimension `Hnz` and type default `INTEGER`. On exit, `Hcol` contains the column indices of the possibly nonzero entries of the approximated Hessian.

`group` is a scalar `INTENT(IN)` argument of type default `LTS_col_groups` (see Subsection E.2.2). It holds all the data on the Hessian structure needed for its approximation.

`permuted` is a scalar `INTENT(IN)` argument of type default `LOGICAL`, that indicates if a permutation is provided in the argument `group`, and must then be used.

`GRAD` is a variable whose value is the name of a user-supplied subroutine whose purpose is to compute the first derivatives of the objective function. See Section E.2.4 for details. It must be declared `EXTERNAL` in the calling program, or an interface must be defined.

E.2.3.2 The column grouping subroutine

The Curtis-Powell-Reed algorithm for gathering non overlapping columns of a Jacobian matrix is called as follows:

```
CALL LTS_cpr_group(n, Hnz, srow, scol, w, u, sym, p, prow, &
    pcol, pinv)
```

`n` is a scalar `INTENT(IN)` argument of type default `INTEGER`, that gives the problem size.

`Hnz` is a scalar `INTENT(IN)` argument of type default `INTEGER`, that gives the number of possibly nonzero entries in the (Jacobian) matrix to reconstruct.

`srow` is an array `INTENT(IN)` argument of dimension `n` + 1 and type default `INTEGER`, that is used with argument `scol` to give the sparsity pattern of the Jacobian matrix in a CSC way. It references the first entry of each column.

`scol` is an array `INTENT(IN)` argument of dimension `Hnz` and type default `INTEGER`, that is used with variable `srow` to give the sparsity pattern of the Jacobian matrix in a CSC way. It holds the row indices of the possibly nonzero entries (sorted by increasing column index).

`u` is an array `POINTER` argument of dimension `n` and type default `INTEGER`, that gives the symmetric permutation used on the Jacobian matrix structure to reduce the number of gradient evaluations required for its approximation ($H(i,j) = H_{\text{permuted}}(u(i), u(j))$).

`w` is an array `POINTER` argument of dimension `n` and type default `INTEGER`, that gives the inverse symmetric permutation ($H(w(i), w(j)) = H_{\text{permuted}}(i,j)$).

`sym` is a scalar `INTENT(IN)` argument of type default `LOGICAL`, that indicates whether a symmetric matrix storage is used (in which case, only the lower triangular part of the matrix stored).

`p` is a scalar `INTENT(OUT)` argument of type default `INTEGER`. On exit, it gives the number of column groups that are used for the Jacobian matrix approximation.

`prow` is an array `POINTER` argument of dimension `p` + 1 and type default `INTEGER`, that is used with argument `pcol` to store the column groups in a CSR way. On exit, it references the first column of each group.

`pcol` is an array `POINTER` argument of dimension `n` and type default `INTEGER`, that is used with variable `prow` to store the column groups in a CSR way. On exit, it holds the column indices gathered by group.

`pinv` is an array `POINTER` argument of dimension `n` and type default `INTEGER`. On exit, it indicates the group to which each column belongs.

E.2.3.3 The symmetric permutation subroutine

The algorithm to construct a symmetric row/column permutation of matrix is called as follows:

```
CALL LTS_permute(n, srow, scol, w, u)
```

`n` is a scalar `INTENT(IN)` argument of type default `INTEGER`, that gives the problem size.

`srow` is an array `POINTER` argument of dimension `n` + 1 and type default `INTEGER`, that is used with argument `scol` to give the sparsity pattern of the matrix in a CSR way. It references the first entry of each row.

`scol` is an array `POINTER` argument of type default `INTEGER`, that is used with variable `srow` to give the sparsity pattern of the matrix in a CSR way. It holds the column indices of the possibly nonzero entries (sorted by increasing row index).

`u` is an array `POINTER` argument of dimension `n` and type default `INTEGER`. On exit, it gives the symmetric permutation used on the matrix structure to reduce the number of gradient evaluations required for its approximation ($H(i,j) = H_{\text{permuted}}(u(i), u(j))$).

`w` is an array `POINTER` argument of dimension `n` and type default `INTEGER`. On exit, it gives the inverse symmetric permutation ($H(w(i), w(j)) = H_{\text{permuted}}(i,j)$).

E.2.3.4 The structure converting subroutine

The algorithm that converts a partial separability structure to a sparsity structure is called as follows:

```
CALL LTS_partSepa2sparse(n, nbrelem, xelvar, elvar, nnz, &
    srow,scol)
```

`n` is a scalar `INTENT(IN)` argument of type default `INTEGER`, that gives the problem size.

`nbrelem` is a scalar `INTENT(IN)` argument of type default `INTEGER`, that gives the number of elements of the partial separability structure.

`xelvar` is an array `POINTER` argument of dimension `nbrelem` + 1 and type default `INTEGER`, that is used with argument `elvar` to hold the partial separability structure in a CSR way. It references the first variable of each element.

`elvar` is an array `POINTER` argument of type default `INTEGER`, that is used with argument `xelvar` to hold the partial separability structure in a CSR way. It holds the variable indices (gathered by element).

`nnz` is a scalar `INTENT(OUT)` argument of type default `INTEGER`. On exit, it gives the number of possibly nonzero entries in the sparsity structure.

`srow` is an array `POINTER` argument of dimension `n` + 1 and type default `INTEGER`, that is used with argument `scol` to store the sparsity pattern of the matrix in a CSR way. On exit, it references the first entry of each row.

`scol` is an array `POINTER` argument of type default `INTEGER`, that is used with variable `srow` to store the sparsity pattern of the matrix in a CSR way. On exit, it holds the column indices of the possibly nonzero entries (sorted by increasing row index).

E.2.3.5 The triangular part extraction subroutine

The algorithm that extracts the lower/upper triangular part of a matrix sparsity structure is called as follows:

```
CALL LTS_lowerpart(n, nnz, srow, scol, u, lnz, lrow, lcol, unz, &
    urow, ucol)
```

`n` is a scalar `INTENT(IN)` argument of type default `INTEGER`, that gives the problem size.

`nnz` is a scalar `INTENT(IN)` argument of type default `INTEGER`, that gives the number of possibly nonzero entries in the matrix.

`srow` is an array `POINTER` argument of dimension `n` + 1 and type default `INTEGER`, that is used with argument `scol` to store the sparsity pattern of the matrix in a CSR way. It references the first entry of each row.

`scol` is an array `POINTER` argument of type default `INTEGER`, that is used with variable `srow` to store the sparsity pattern of the matrix in a CSR way. It holds the column indices of the possibly nonzero entries (sorted by increasing row index).

`u` is an array `INTENT(OUT)` argument of dimension `n` and type default `INTEGER`, that gives the symmetric permutation used on the matrix structure to reduce the number of gradient evaluations required for its approximation ($H(i,j) = H_{\text{permuted}}(u(i), u(j))$).

`lnz` is a scalar `INTENT(OUT)` argument of type default `INTEGER`, that gives, on exit, the number of possibly nonzero entries in the lower triangular part of the matrix.

`lrow` is an array `POINTER` argument of dimension `n` + 1 and type default `INTEGER`, that is used with argument `lcol` to store the sparsity pattern of the lower triangular part of the matrix in a CSR way. On exit, it references the first entry of each row.

`lcol` is an array `POINTER` argument of type default `INTEGER`, that is used with variable `lrow` to store the sparsity pattern of the lower triangular part of the matrix in a CSR way. On exit, it holds the column indices of the possibly nonzero entries (sorted by increasing row index).

`unz` is a scalar `INTENT(OUT)` argument of type default `INTEGER`, that gives, on exit, the number of possibly nonzero entries in the upper triangular part of the matrix.

`urow` is an array `POINTER` argument of dimension `n` + 1 and type default `INTEGER`, that is used with argument `ucol` to store the sparsity pattern of the upper triangular part of the matrix in a CSR way. On exit, it references the first entry of each row.

`ucol` is an array `POINTER` argument of type default `INTEGER`, that is used with variable `urow` to store the sparsity pattern of the upper triangular part of the matrix in a CSR way. On exit, it holds the column indices of the possibly nonzero entries (sorted by increasing row index).

E.2.3.6 The optimal grouping subroutine

The algorithm that returns the optimal grouping for predefined Hessian structure is called as follows:

```
CALL LTS_getOptimalGroups(n, groups, hnz, sparsity_type)
```

`n` is a scalar `INTENT(IN)` argument of type default `INTEGER`, that gives the problem size.

`groups` is a scalar `INTENT(OUT)` argument of type default `LTS_col_groups` (see Subsection E.2.2). It holds all the data on the Hessian structure needed for its approximation.

`hnz` is a scalar `INTENT(OUT)` argument of type default `INTEGER`, that gives the number of possibly nonzero entries in the Hessian matrix.

`sparsity_type` is a scalar `INTENT(IN)` argument of type default `INTEGER`, that indicates a particular sparsity pattern for which an optimal column grouping is already known. We refer to Section 8.3 for a list of the allowed values and the description of the corresponding patterns.

E.2.4 Evaluating the first derivatives of problem functions

The first derivatives of the the objective function (f) must be provided by the user as a subroutine, with a prescribed argument list, that accepts input values (x), and returns as output $\nabla f(x)$. If one uses `GRAD = MY_GRAD` in the call to `LTS_powetoin`, the `MY_GRAD` routine must have the following argument list:

```
SUBROUTINE MY_GRAD(x, g)
```

where

`x` is an array `INTENT(IN)` argument of dimension n and type default `REAL` (double precision in `GALAHAD_LTS_double`), that contains the point x at which the subroutine is required to evaluate the gradient of the objective function f.

`g` is an array `INTENT(OUT)` argument of dimension n and type default `REAL` (double precision in `GALAHAD_LTS_double`), that contains the value of the gradient evaluated at x.

E.2.5 Warning and error messages

Every error messages is sent to the console. The subroutines `LTS_powetoin`, `LTS_cpr_group`, `LTS_permute`, `LTS_partSepa2sparse`, `LTS_lowerpart`, `LTS_-getOptimalGroups` may terminate with an allocation error, while the latter subroutine may also terminate if an unknown sparsity type is required.

E.3 General information

Use of common: None.

Workspace: Provided automatically by the module

Other routines called directly: `GRAD`, a user-supplied routine computing the gradient of the function whose Hessian is to be approximated.

Other modules used directly: None.

Input/output: Normally, no output in console.

Restrictions: $n > 0$.

Portability: ISO Fortran 95 + TR 15581 or Fortran 2003. The package is thread-safe.

E.4 Example of use

Suppose we wish to approximate the Hessian of the function $f : x \mapsto \frac{1}{2}x^T Hx$, whose Hessian is the matrix

$$H = \left(\begin{array}{ccc|ccc|ccc} 4 & -1 & & -1 & & & & & \\ -1 & 4 & -1 & & -1 & & & & \\ & -1 & 4 & & & -1 & & & \\ \hline -1 & & & 4 & -1 & & -1 & & \\ & -1 & & -1 & 4 & -1 & & -1 & \\ & & -1 & & -1 & 4 & & & -1 \\ \hline & & & -1 & & & 4 & -1 & \\ & & & & -1 & & -1 & 4 & -1 \\ & & & & & -1 & & -1 & 4 \end{array}\right)$$

Then we may use the following code

```
PROGRAM GALAHAD_LTS_EXAMPLE

  USE GALAHAD_LTS_DOUBLE   ! double precision version

  IMPLICIT NONE

  INTEGER, PARAMETER :: wp = KIND(1.0D+0)

  INTEGER :: i, siz, Hnz
```

```
  INTEGER,       DIMENSION(:), POINTER :: Hrow, Hcol
  REAL(KIND=wp), DIMENSION(:), POINTER :: x
  REAL(KIND=wp), DIMENSION(:), POINTER :: Hval
  TYPE(LTS_col_groups ) :: groups

  EXTERNAL :: MY_GRAD

  siz = 3 ** 2;
  ALLOCATE(x(siz))

  ! Get optimal column grouping for Laplacian
  CALL LTS_getOptimalGroups(siz, groups, Hnz, 1)

  ! Compute the approximate Hessian in COO format
  CALL LTS_powetoin(siz, x, Hnz, Hval, Hrow, Hcol, groups, &
       .FALSE., MY_GRAD)

  WRITE (*, 12) groups%p+1

  DO i = 1, Hnz
     WRITE(*, 10) Hrow(i), Hcol(i), Hval(i)
  END DO

  DEALLOCATE(x, Hval, Hrow, Hcol)

10 FORMAT(' H(',i1,',',i1,') = ',f16.12)
12 FORMAT('Hessian computation has required ', i2, &
        ' gradient evaluations')

END PROGRAM GALAHAD_LTS_EXAMPLE

SUBROUTINE MY_GRAD(x, g)

  ! Gradient of the quadratic function x -> x^T*H*x / 2
  ! where H is a Laplacian matrix like this:
  !            (  4 -1    -1                 )
  !            ( -1  4 -1    -1              )
  !            (    -1  4       -1           )
  !            ( -1         4 -1    -1        )
  !            (    -1    -1  4 -1    -1     )
  !            (       -1    -1  4       -1  )
  !            (          -1       4 -1     )
  !            (             -1    -1  4 -1  )
  !            (                -1    -1  4  )

  IMPLICIT NONE

  INTEGER, PARAMETER :: wp = KIND(1.0D+0)

  REAL(KIND=wp), DIMENSION(:), INTENT(IN ) :: x
  REAL(KIND=wp), DIMENSION(:), INTENT(OUT) :: g

  INTEGER :: i, j, k, n, z
  REAL(KIND=wp), PARAMETER :: FOUR = 4.0_wp
```

```
  n = SIZE(x)            ! assume to be square
  z = INT(SQRT(DBLE(n)))

  i = 1
  DO k = 1, z
     DO j = 1, z
        g(i) = FOUR * x(i)
        IF (k > 1) g(i) = g(i) - x(i-z)
        IF (j > 1) g(i) = g(i) - x(i-1)
        IF (j < z) g(i) = g(i) - x(i+1)
        IF (k < z) g(i) = g(i) - x(i+z)
        i = i + 1
     END DO
  END DO

END SUBROUTINE MY_GRAD
```

This produces the following output:

```
Hessian computation has required  4 gradient evaluations
 H(9,6) =  -1.000000000000
 H(6,9) =  -1.000000000000
 H(9,8) =  -1.000000000000
 H(8,9) =  -1.000000000000
 H(9,9) =   4.000000000000
 H(8,5) =  -1.000000000000
 H(5,8) =  -1.000000000000
 H(8,7) =  -1.000000000000
 H(7,8) =  -1.000000000000
 H(8,8) =   4.000000000000
 H(7,4) =  -1.000000000000
 H(4,7) =  -1.000000000000
 H(7,7) =   4.000000000000
 H(6,3) =  -1.000000000000
 H(3,6) =  -1.000000000000
 H(6,5) =  -1.000000000000
 H(5,6) =  -1.000000000000
 H(6,6) =   4.000000000000
 H(5,2) =  -1.000000000000
 H(2,5) =  -1.000000000000
 H(5,4) =  -1.000000000000
 H(4,5) =  -1.000000000000
 H(5,5) =   4.000000000000
 H(4,1) =  -1.000000000000
 H(1,4) =  -1.000000000000
 H(4,4) =   4.000000000000
 H(3,2) =  -1.000000000000
 H(2,3) =  -1.000000000000
 H(3,3) =   4.000000000000
 H(2,1) =  -1.000000000000
 H(1,2) =  -1.000000000000
 H(2,2) =   4.000000000000
 H(1,1) =   4.000000000000
```